Australia A $4.75 New Zealand NZ $ 6.50
Singapore S $ 9.45 Hong Kong H $23.50
Malaysia M $ 9.45 Sweden 30 SEK

$2.95 USA

68' MICRO JOURNAL

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems
6809 68008 68000 68010 68020 68030
OS-9 The Magazine for Motorola CPU Devices FLEX SK*DOS
A User Contributor Journal

This Issue:

And Lots More!

VOLUME IX ISSUE VII ● Devoted to the 68XXX User ● July 1987

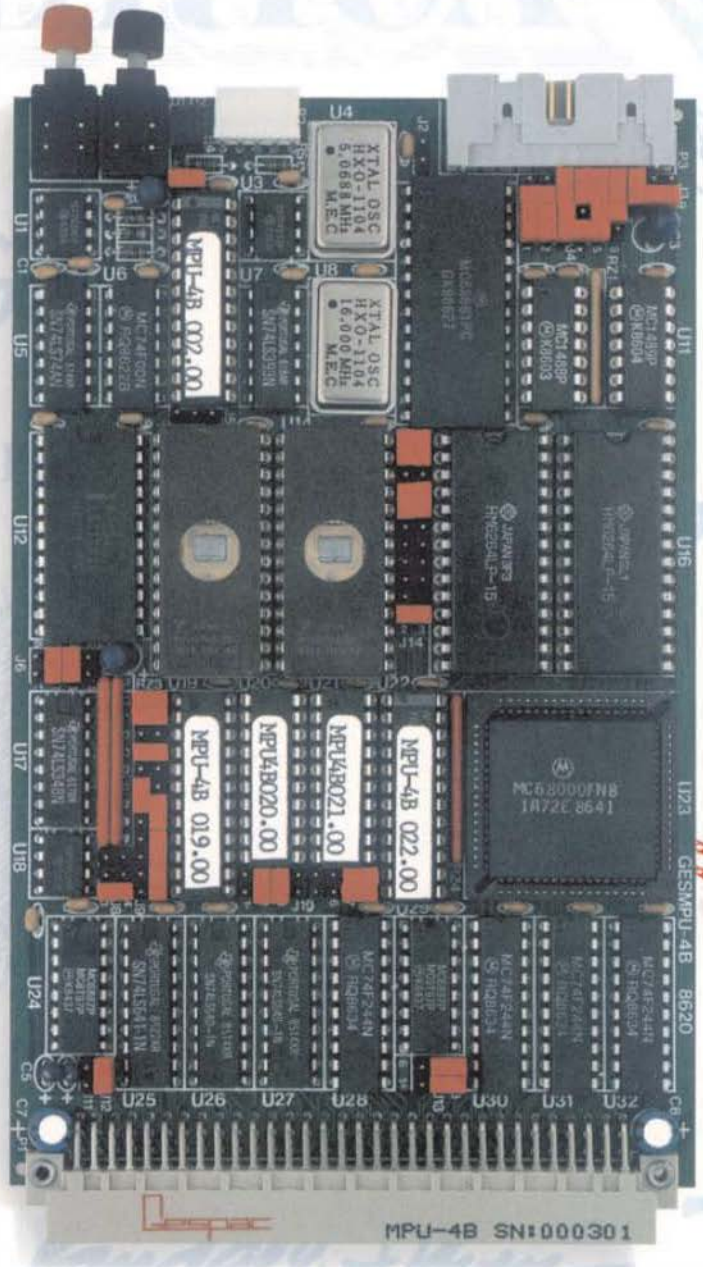
The "Grandfather of Desk Top Publishing"

SERVING THE 68XXX USER WORLDWIDE

PHOTO CREDIT: NASA

0 74470 12810 07

GESPAC Gives You 68000 Performance at 8-bit Prices.



Actual Size

Introducing another Price / Performance breakthrough from GESPAC: A complete 68000 CPU module for $395 unit price. The GESMPU-4B.

Wherever you thought of using on 8-bit microprocessor to do a job, you con now use the 16/32-bit GESMPU-4B instead. It will do the job, better, foster, and best-of-all for the same money.

On a single height Eurocard, we hove pocked on 8 MHz 68000 microprocessor (16 MHz optional), four sockets for up to 64 Kilobytes of zero-wait-states CMOS RAM and up to 128 Kilobytes of EPROM, one RS-232 serial port, and three 16-bit timers.

The GESMPU-4B is fully expandable through the standard G-64 bus, to accommodate up to 16 Megabytes of external memory. You con odd any of more than 300 I/O modules available from GESPAC and a growing numbers of independent G-64 bus vendors.

To make your programming tasks easier, GESPAC supports the GESMPU-4B with the OS-9® and PDOS® real-time, multi-tasking operating systems and most popular high level languages and software development tools.

$395.00
Single Quantity

$316 @ 100 pieces

If you too like the ideo of getting more for less, contact us today to receive information on the GESMPU-4B and the G-64 bus concept from GESPAC—the leader in single Eurocard microcomputer products worldwide.

Call Toll Free 1-800-4-GESPAC
or Call (602) 962-5559.

Gespac

IN USA - CANADA
50A West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Telex 386575

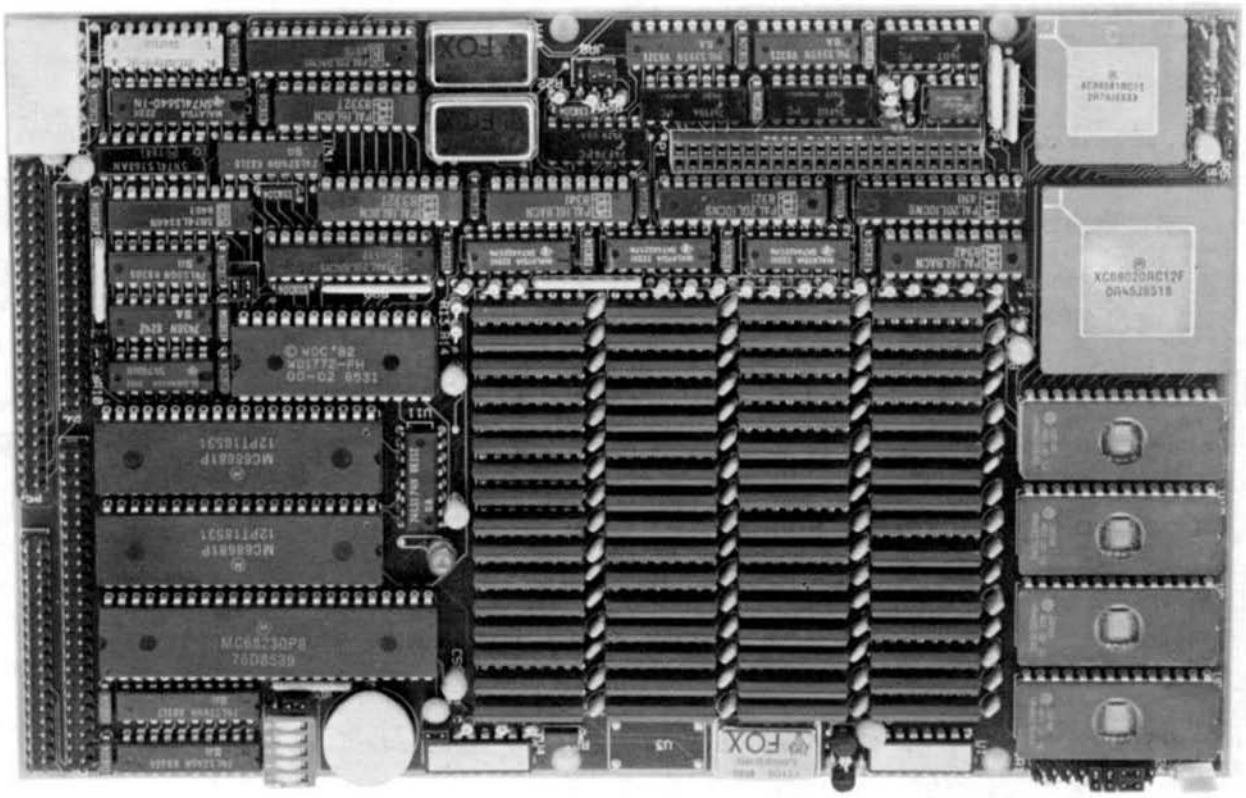
INTERNATIONAL
3, chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

GMX™ Micro-20
68020 SINGLE—BOARD COMPUTER

Mainframe CPU Performance on a 5.75" x 8.8" Board
(benchmark results available on request)

$2565^{00}

12.5 MHz Version
Quantity Discounts Available



Features

- 32-Bit MC68020 Processor (12.5, 16.67, or 20MHZ)
- MC68881 Floating-point coprocessor (optional)
- 2 Megabytes of 32-bit wide, high-speed RAM
- 4 RS-232 Serial I/O Ports (expandable to 36)
- 8-bit Parallel I/O Port ('Centronics' compatible)
- Time-of-Day Clock w/battery backup
- 16-bit I/O Expansion Bus
- Up to 256 Kbytes of 32-bit wide EPROM
- Floppy Disk Controller for two 5¼" drives
- SASI Intelligent Peripheral Interface (SCSI subset)
- Mounts directly on a 5¼" Disk Drive
- Optional Boards include Arcnet, Prototyping, I/O Bus adapter, 60 line Parallel I/O, RS-422/485

Software

Included:

- GMX Version of Motorola's 020Bug Debugger with up/download, breakpoint, trace, single-step, and assembler/disassembler capabilities
- Comprehensive Hardware Diagnostics

Optional:

UNIX™ like Multi-user/Multi-tasking Disk Operating Systems

- OS-9/68000™ (Real-time and PROMable)
- UniFLEX™

Programming Languages and Application Software

- BASIC, C, PASCAL, ABSOFT FORTRAN, COBOL and ASSEMBLER
- Spreadsheet, Data Base Management, and Word Processing

COMPLETE EVALUATION SYSTEMS AVAILABLE

GMX INC. 1337 W. 37th Place Chicago, IL 60609

(312) 927-5510 • TWX 910-221-4055
State-of-the-Art Computers Since 1975

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

The VME BUS and OS-9:

Ultimate Software for the Ultimate Bus.

Modularity. Flexibility. High Performance. Future growth. These are probably the prime reasons you chose the VME bus. Why not use the same criteria when selecting your system software? That's why you should take a look at Microware's OS-9/68000 Operating System—it's the perfect match for the VME bus.

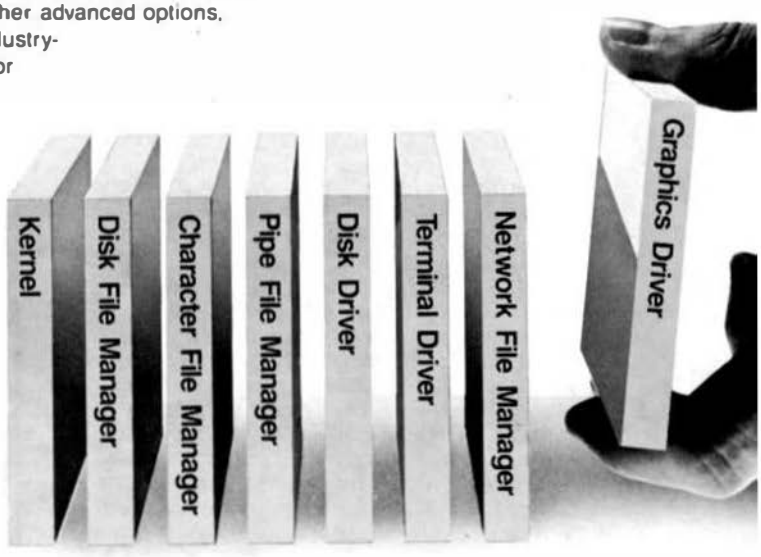
When you're working with VME you must have access to every part of the system. Unlike other operating systems that literally scream KEEP OUT!, OS-9's open architecture invites you to create, adapt, customize and expand. Thanks to its unique modular design, OS-9 naturally fits virtually any system, from simple ROM-based controllers up to large multiuser systems.

And that's just the beginning of the story. OS-9 gives you a complete UNIX-application compatible environment. It is multitasking, real time, and extremely fast. And if you're still not impressed, consider that a complete OS-9 executive and I/O driver package typically fits in less than 24K of RAM or ROM.

Software tools abound for OS-9, including outstanding Microware C, Basic, Fortran, and Pascal compilers. In addition, cross C compilers and cross assemblers are available for VAX systems under Unix or VMS. You can also plug in other advanced options, such as the GSS-DRIVERS™ Virtual Device Interface for industry-standard graphics support, or the OS-9 Network File Manager for high level, hardware-independent networking.

Designed for the most demanding OEM requirements, OS-9's performance and reliability has been proven in an incredible variety of applications. There's nothing like a track record as proof: to date, over 200 OEMs have shipped more than 100,000 OS-9-based systems.

Ask your VME system supplier about OS-9. Or you can install and evaluate OS-9 on your own custom system with a reasonably priced Microware PortPak™. Contact Microware today. We'll send you complete information about OS-9 and a list of quality manufacturers who offer off-the-shelf VME/OS-9 packages.



Modular Hardware Deserves Modular Software

MICROWARE.

Microware Systems Corporation
1866 N.W. 114th Street • Des Moines, Iowa 50322
Phone 515-224-1929 • Telex 910-520-2535

Microware Japan, Ltd.
41-19 Honcho 4-Chome, Funabashi City • Chiba 273, Japan • Phone 0473 (28) 4493 • Telex 781-299-3122

Micromaster Scandinavian AB
S:t Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
Phone: 018-138595
Telex: 76129

Dr. Rudolf Keil, GmbH
Porphystrasse 15
D-6905 Schriesheim
West Germany
Phone: (0 62 03) 67 41
Telex: 465025

Elsoft AG
Zelweg 12
CH-5405 Baden-Dattwil
Switzerland
Phone: (056) 83-3377
Telex: 828275

Vivaway Ltd.
36-38 John Street
Luton, Bedfordshire, LU1 2JE
United Kingdom
Phone: (0582) 423425
Telex: 825115

Microprocessor Consultants
92 Bynya Road
Palm Beach 2108
NSW Australia
Phone: 02-919-4917

Microdata Soft
97 bis, rue de Colombes
92400 Courbevoie
France
Phone: 1-768-80-80
Telex: 615405

OS-9 is a trademark of Microware and Motorola. PortPak is a trademark of Microware. GSS-Drivers is a trademark of Graphic Software Systems, Inc. VAX and VMS are trademarks of DEC. Unix is a trademark of AT&T.

MUSTANG -020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super nucros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP
Installed Systems World-Wide
OVER 10 YEARS OF DEDICATED QUALITY!
CPI A Division of Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9
68XXX
Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED **SCREEN ORIENTED TEXT EDITOR** with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO
PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter begins the discussion of the conversion of Technical Systems Consultants BASIC and Microware BASIC09 programs into C programs. It suggests methods which may be used to approach the more non-trivial conversion problems. It does not provide direct conversion algorithms, nor does it address the media conversion problems which may be encountered when changing operating systems. Most of the problems discussed are indigenous to the conversion of BASIC programs to C, rather than being version-specific, so much of the discussion would apply to other versions of BASIC.

CONVERTING BASIC PROGRAMS TO C

There are two versions of TSC Extended BASIC: the pre-compiler (XPC) and the interpreter (XBASIC). The pre-compiler is a tokenizer and is capable of processing much larger programs than will fit into memory (especially on 16-bit machines) with the BASIC interpreter. It also extends the BASIC language in certain manners, most notably in terms of longer variable names, generalized statement continuation, non-numeric labels, and the elimination of the requirement that every line be numbered.

Even if the pre-compiler is used, the interpreter is required to process the tokenized version of the program. The interpreter also capable of tokenizing and saving a program which has been loaded.

There is only one version of the Microware BASIC09 interpreter. It always tokenizes its source programs. This product is somewhat misnamed, in that in many ways it is more like a Pascal interpreter than a BASIC interpreter. Unfortunately, some of these differences make the conversion of BASIC09 programs to C much more challenging.

In addition to the BASIC09 interpreter, there is a run-time-only interpreter, called BRUN, which only interprets pre-tokenized files. Both versions of the interpreter are capable of linking to OS-9 modules (which need not be BASIC09 modules) and are capable of unlinking from already-linked OS/9 modules, so very large programs may be run and/or interpreted, in a modular fashion.

Variable Names

XBASIC allows a variable name to be composed of one or two characters, the first of which must be a letter, and the second of which may be a letter or digit. This follows the original Dartmarth BASIC standard for variable naming conventions.

XPC allows a variable name of one to 255 characters, the first of which must be a letter, and the remainder of which may be letters, digits, or underlines.

TSC BASIC allows a variable name to be followed by a dollar sign or percentage sign, indicating that the variable name is not to represent a floating-point number, which BASIC variables normally represent by default, but is to represent a string or 16-bit integer number, respectively.

BASIC09 allows a variable name of one to 32 characters, the first of which must be a letter, and the remainder of which may be digits. It allows a variable name to be followed by a dollar sign, indicating that the variable is to represent a string, rather than a floating point number. BASIC09 also supports a type declaration which allows the specification of a variable type as byte (one unsigned character), integer (16-bit field), real (5-byte floating-point), string (adjacent-character field), boolean (true or false). In the case of undeclared strings, the maximum length is fixed at 32 bytes.

The TSC BASIC interpreters fold upper and lower case letters in reserved words, function names, and in variable names. Thus "READ" is the same as "read", "SIN" is the same as "sin", and "A1" is the same as "a1". BASIC09 folds upper and lower case in reserved words and function names, but not in variable names, so that "READ" is the same as "read", "SIN" is the same as "sin", but "A1" is different from "a1".

C compilers require variable names to start with a letter or underline, and to be composed of letters, digits, and underlines. They are always sensitive to case for all types of internal variable names. The rules for linking external variables across modules vary by system.

All of the BASICs discussed here allow variables to be subscripted, with one or two (three for BASIC09) dimensions, through the use of the DIM statement. TSC BASIC extends this concept with virtual arrays, which provide a notational shorthand for random disk files.

The first element of a TSC BASIC subscripted variable has index zero. The first element of a BASIC09 subscripted variable has index one by default, although the BASE statement may be used to set this default value to zero or one explicitly. In practice, this difference between BASIC and C may normally be safely ignored, perhaps at the expense of not using subscript zero, since C array subscripting assumes base zero.

TSC BASIC initializes floating-point ant integer variables to zero and non-virtual strings to null. BASIC09 initializes no variables. C compilers initialize global variables to zero, but auto variables are not initialized to any particular values.

C requires different notation for multiple-dimensioned variables than does BASIC, as follows:

```
v(n)          v[n]
v(n1,n2)      v[n1][n2]
v(n1,n2,n3)   v[n1][n2][n3]
```

No variable name in any version of BASIC may be in a list of reserved words for that version of BASIC. However, these BASIC variable names may very well conflict with reserved C words or otherwise violate the rules of formation of C variable names (in terms of length, for instance), the user must be wary of such naming conflicts. This situation usually results in syntax errors, but may occasionally cause more unusual problems, such as two undeclared BASIC variables effectively becoming one C variable, etc. Those BASIC variable names that are illegal in C should be changed before conversion is attempted, the exceptions being dollar sign and percentage sign, which must be changed during conversion, since they may specify the C variable type.

Since all variables must be declared in C, all the BASIC variables must be located. TSC BASIC variables are all global, but BASIC09 variables may be considered local to the PROCEDURE in which they are declared. For the purpose of the translation, local BASIC09 variables may often be considered the same as global. However, conflicting names across functions and recursive functions may complicate this assumption in some cases.

Since BASIC interpreters allow simple and subscripted variables and floating-point, integer, and string variables to have the same names, a trailing character depending upon the data type and subscript status may be necessary to ensure name uniqueness.

In BASIC09, parameters are usually passed by reference to other PROCEDUREs, but they may be passed by value by making them part of an expression. In C, parameters are always passed by value. A value may be returned to a calling function by passing the address of a variable, not the variable itself, although this is not an entirely satisfactory solution to the problem. In many cases, treating all BASIC09 variables and parameters as global may effectively solve the variable declaration problem. In other cases, some combination of global local variables may be appropriate.

Statements and Labels

XBASIC requires that every statement be labelled with an integer in the range of 1 to 32767. It maintains them in ascending order. Multiple statements may appear on the same line, separated by colons or reverse slashes; however, only the first statement of the group may be labelled. A single line may not contain more than 127 characters.

XPC requires only those statements used in GOSUB, GOTO, RESTORE, and RESUME statements and in expressions involving ERL to be labelled, although other lines may be labelled, and allows labels to be integers or to follow the same rules as XPC variables. It requires labels to start in the first character position. Multiple statements may appear on the same line, separated by colons or reverse slashes; however, only the first statement of the group may be labelled. A physical line may not contain more than 255 characters; however, a line may be continued with a colon or reverse slash and carriage return sequence, so the logical line length is not limited to 255 characters.

In either version of TSC BASIC, an IF statement applies to the remainder of the statements to its right on the logical line.

BASIC09 requires labels to be integers, but does not require every statement to be labelled. Multiple statements may appear on the same line, separated by reverse slashes; however, only the first statement of the group may be labelled. A single line may not contain more than 255 characters. Line feeds may be used to separate sections of a long statement, rather than carriage returns, for listing format purposes.

The C compilers have no such formatting restrictions. Labels may start in any character position within a line, but must start with a letter

or underline and must be followed by a colon. In most C compilers, a logical line may be continued by preceding the terminating new line character with a back-slash.

Strings and Numbers

TSC BASIC supports strings of length 0 to 32767 bytes with arbitrary contents, a current length, and dynamic allocation. It also supports string arrays of one or two dimensions. A string constant is delimited by single or double quote characters but may not contain the same type of quote character as the delimiters.

BASIC09 supports strings of length 0 to 65535 bytes with contents of all bytes except $FF (the terminator), a maximum length, and fixed allocation. It also supports string arrays of up to three dimensions. The maximum length of undeclared strings is 32 characters. String constants are delimited by double quote characters and may contain a double quote character by placing two adjacent double quote characters, representing one double quote character.

TSC BASIC supports the following numeric types and arrays of up to two dimensions based on these types:

```
integer
    -32768 to +32767
    (2 bytes)
floating point
    17 digits
    (8 bytes)
```

BASIC09 supports the following numeric types and arrays of up to three dimensions based on these types:

```
boolean
        TRUE or FALSE
        (1 byte)
byte
        0 to 255
        (1 byte)
integer
        -32768 to +32767
        (2 bytes)
hex constant (declared integer)
        $0000 to $ffff
        (2 bytes)
floating point
        9 digits
        (5 bytes)
```

Although the lengths are not universally implemented, C generally supports the following types and arrays of any dimension based on these types:

```
signed char
        -128 to +127
        (1 byte)
unsigned char
        0 to 255
        (1 byte)
signed short int or int
        -32768 to +32767
        (2 bytes)
unsigned short int or int
        0 to 65535
        (2 bytes)
signed int or long
        -2147483648 to +2147483647
        (4 bytes)
unsigned int or long
        0 to 4294967295
        (4 bytes)
float
        8 digits
        (4 bytes)
double
        17 digits
        (8 bytes)
```

BASIC09 also supports complex data types comprised of one-dimensional vectors of multiple elementary data types (boolean, byte, integer, floating-point, and string). These structures are highly restricted in terms of legal operations on them. For example, two structures may not be added together, although numeric elementary data types within two structures may be added together. Structures may be moved and read or written as a unit, making them very useful in such cases.

TSC BASIC strings carry a true length and may have arbitrary contents, while BASIC09 strings have a maximum length and require the concatenation of an hex-ff character code to indicate a shorter string.

TSC BASIC and BASIC09 provide temporary string storage areas, string space garbage collection, and other facilities to materially simplify string processing, such as assignment and concatenation, plus string functions which themselves require temporary string space.

The C compilers leave all such string processing to the programmer. C strings have a maximum length and usually are terminated with a hex-00 character. These differences in processing philosophy among the BASICs and the C compilers will increase the difficulty of the conversion materially in many cases.

Although all floating-point BASIC variables may be mapped into double C variables, they may in many cases be safely mapped into float, long, int, short int, or char variables, to save space and time. Only careful analysis of each the usage of each BASIC variable can determine if such optimizations are possible.

This discussion is continued in the next chapter.

EXAMPLE C PROGRAM

Following is this month's example C program; it compares two text files and outputs the differences between them.

```
#include <stdio.h>
#include <ctype.h>
#define DEPTH 101
#define LRECL 80

FILE *oldfile, *newfile;
char a[DEPTH][LRECL], b[DEPTH][LRECL], ae, be,
ef[2], *p, *q;
int depth = 10, an[DEPTH], bn[DEPTH];
int al = -1, bl = -1, a2, b2, a3, b3, c, d1,
am, bm, dbg;
int i, j, tens[5] = {10000, 1000, 100, 10, 1};

main(argc, argv)
int argc;
char *argv[];
{
    putc('\n', stdout);
    if (dbg = (*argv[1] == '-'))
        ++argv, --argc;
    if (argc < 3)
    {
        fputs("Usage: ", stderr);
        fputs(argv[0], stderr);
        fputs(" old-file new-file
[depth]\n\nwhere:\n", stderr);
        fputs(" old-file is old version of
file\n", stderr);
        fputs(" new-file is new version of
file to be compared\n", stderr);
        fputs(" depth is size of look-ahead
table (10-100)\n\n", stderr);
        exit(1);
    }
    if (!(oldfile = fopen(argv[1], "r")))
    {
        fputs("Could not open old file ",
stderr);
        fputs(argv[1], stderr);
        putc('\n', stderr);
        exit(1);
    }
    if (!(newfile = fopen(argv[2], "r")))
    {
        fputs("Could not open new file ",
stderr);
        fputs(argv[1], stderr);
        putc('\n', stderr);
        exit(1);
    }
    if (argc > 3)
    {
        for (depth = 0, p = argv[3]; *p; ++p)
            if (isdigit(*p))
                depth = depth * 10 + (*p -
'0');
        if (depth < 10)
            depth = 10;
        else
        if (depth >= DEPTH)
            depth = DEPTH - 1;
    }
    *ef = 0xff;
    readfile('o', oldfile, a, &ae, &am, an,
&a2);
    readfile('n', newfile, b, &be, &bm, bn,
&b2);
    a2 = b2 = 1;
getold:
    getfile('o', oldfile, a, &ae, &am, an,
&al, &a2);
getnew:
    getfile('n', newfile, b, &be, &bm, bn,
&bl, &b2);
    if (!strcmp(a[al], b[bl]))
        if (strcmp(a[al], ef))
            goto getold;
        else
        {
            fclose(oldfile);
            fclose(newfile);
            exit(0);
        }
    a3 = al;
    b3 = bl;
    d1 = 0;
nextab:
    if (++d1 >= depth)
    {
        if (strcmp(a[al], ef))
        {
            fputs("--*o", stdout);
            outnumb(an[al]);
            fputs(a[al], stdout);
            putc('\n', stdout);
        }
        if (strcmp(b[bl], ef))
        {
            fputs("++*n", stdout);
            outnumb(bn[bl]);
            fputs(b[bl], stdout);
            putc('\n', stdout);
        }
        goto getold;
    }
    a3 = (a3 >= depth) ? 0 : a3 + 1;
    b3 = (b3 >= depth) ? 0 : b3 + 1;
    if (!strcmp(b[bl], a[a3]))
    {
        do
        {
```

```
            if (strcmp(a[a1], ef))
            {
                fputs("-- o", stdout);
                outnumb(an[a1]);
                fputs(a[a1], stdout);
                putc('\n', stdout);
            }
            a1 = (a1 >= depth) ? 0 : a1 + 1;
        }
        while (a1 != a3);
        goto getold;
    }
    if (!strcmp(a[a1], b[b3]))
    {
        do
        {
            if (strcmp(b[b1], ef))
            {
                fputs("++ n", stdout);
                outnumb(bn[b1]);
                fputs(b[b1], stdout);
                putc('\n', stdout);
            }
            b1 = (b1 >= depth) ? 0 : b1 + 1;
        }
        while (b1 != b3);
        goto getnew;
    }
    goto nextab;
}

readfile(o, file, a, ae, am, an, a2)
FILE *file;
char a[DEPTH][LRECL], *ae;

   int o, *am, an[DEPTH], *a2;
   {
       if (!*ae)
       {
           an[*a2] = ++(*am);
           if (fgets(a[*a2], LRECL, file))
           {
               if ((i = strlen(a[*a2])) &&
(a[*a2][--i] < 0x20))
                   a[*a2][i] = 0;
               if (dbg)
               {
                   putc(o, stdout);
                   outnumb(*a2);
                   outnumb(*am);
                   fputs(a[*a2], stdout);
                   putc('\n', stdout);
               }
               return;
           }
       }
       an[*a2] = *am + (*ae = 1);
       strcpy(a[*a2], ef);
       if (dbg)
       {
           putc(o, stdout);
           outnumb(*a2);
           outnumb(*am);
           fputs("<eof>\n", stdout);
       }
   }

   getfile(o, file, a, ae, am, an, a1, a2)
   FILE *file;
   char a[DEPTH][LRECL], *ae;
   int o, *am, an[DEPTH], *a1, *a2;
   {
       *a1 = (*a1 >= depth) ? 0 : *a1 + 1;
       do
       {
           readfile(o, file, a, ae, am, an, a2);
           *a2 = (*a2 >= depth) ? 0 : *a2 + 1;
       }
       while ((*a1 != *a2) && (!*ae));
   }

   outnumb(n)
   int n;
   {
       putc(' ', stdout);
       for (i = 0; i < 5; ++i)
       {
           for (j = '0'; n >= tens[i]; ++j, n -=
tens[i]);
           putc(j, stdout);
       }
       putc(' ', stdout);
   }
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68030

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

GETTING INTO DIRECTORIES

This was probably the first contact with a system. Any system. There are a lot of things you can do first. But, I'll wager, the first thing you did was enter that special word. Obviously there were a few items to be first accomplished. There was bootup. Or if it were in a multi-user system, a standard login procedure was used. But, then it came...DIR.

The first time out for me (again before my OS-9 days), I asked, "What should I do first?" Ah! My computer advisor made the suggestion. "Read the directory!" So, my fingers typed D-I-R. Whirl! Click-click! went the drive. And there it was. All the secrets, the disk held appeared before me. Programs! Files! Games! And more! All of its contents were there. Then I saw its first failure.

My screen could display 24 lines. My disk contained much more than that amount. So, I learned the pause technique. With deft hands, I could strike the CNTRL-S and freeze the screen. A CNTRL-Q and it would again scroll. It took practice, but I could start and stop screen scrolling, pretty accurately. The drives got bigger. And so did the number of files. The number of starts and stops increased. An intermediate solution came.

It was the wild card.q The wild card offered some relief. Select parts of disk could be viewed. A reasonable facsimile of the file name of interest could be added to DIR. To see all files that started with the letter A, an entry like:

DIR A*

could be used. The star was the wild card. This helped a little. As long as you had some idea of what you wanted to see, this method was helpful. But, it did not answer the question of disk organization. This required an alternate view and some improvements over the older method. A simple solution did it. Put a directory within a directory. This is a solution that has its origins with the large mainframes, but is now common in to the level of PC's. OS-9 allows the creation of directories within directories.

It may be helpful to study how OS-9 creates a directory. Although I have covered some of this before, a quick review can prove to be most helpful. Even if you are not technically oriented a small knowledge of how the directory is stored can be useful. Later, it will throw some light on other features of working with the directory.

First, it should be understood that the directory is only a specialized file. It is assigned a file descriptor sector. There is a small difference exists. Its attribute byte has the most significant bit is set. This sets it apart from any other file. You can delete a file. Just use the DEL command. It won't work on the directory. Imagine the horror of deleting a directory with all the files it contained stuck in limbo! Never to be used again, and never to be removed. This is why exits the command DELDIR. It first deletes all the files in the

directory. Then, the MSB of the attribute is toggled off. This demotes it from directory to a file. Then finally, it is deleted. A rather lengthy process when compared with the simple deleting of a file, but necessary to preserve the integrity of the media where it was stored.

Next to look at is the manner in which the entries are stored. Each entry is 32 bytes long. 29 bytes are reserved for the name. And three bytes are used to find the sector where the file's descriptor is located. The end of the file's name is flagged by setting the MSB of the last byte. Whenever a file is deleted from the directory, the first byte of the entry is set to zero. The directory table contains two very special entries. They are dot and

dot-dot. The dot which is really entered as ".". Points to the current directory. Dot-dot or ".." is the parent directory. So when entering:

DIR .

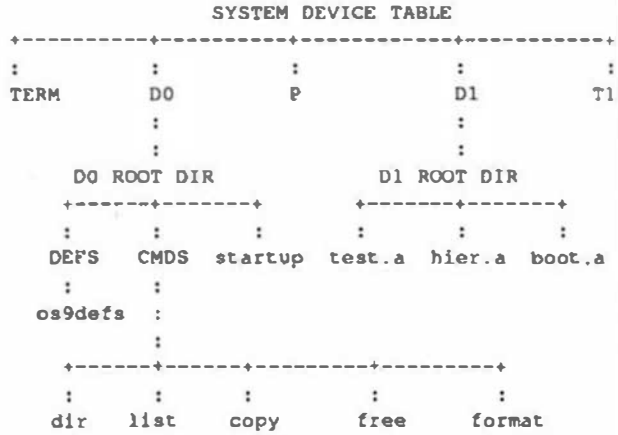
The dot directory is opened and listed. Entering:

DIR ..

reads the previous directory. In fact dots can be put together to read back as far as you like in previous directories. The operating system is capable of backing itself out of any place you may be.

If you have noticed, in the directory table no mention is made whether the entry is a directory or file. Hence, a directory can be located within another directory. Using directory entries and file descriptor sectors, everything is put together. It all starts at some top level. This is some type of storage media. It could be a 5" or 8" disk. Maybe it is a hard disk. But it starts at some top level directory and fans outward.

Now that we understand the physical mechanism of how directories work, let's move on to the actual structures. Everything is organized into tree structured hierarchies. It all starts at OS-9's Device Directory. This is managed by the system. It contains all SCF and RBF devices. The SCF devices like TERM and P are dead ends. You could not build on them. The RBF devices however contain Root Directories. These are tables like I described before. Either they point to files or more directories. If it is a directory, then that directory has its own table of entries that point to more files and directories. This can go on into infinity or until the disk runs out. (I'll bet on the later!) It is best to look at an example.



This describes how a typical system could be set up. The device table contains the SCF devices TERM, P, and T1. They cannot have any directories. But the RBF devices, D0 and D1, have root directories. D1 contains three assembler files. D0 has two directories and a file. Each of those directories have files in them.

This hierarchical directory structure makes organizing files a snap. Directories for specific file types can be created. This becomes a matter of personal taste and needs. There are a few standard ones. The most obvious one is CMDS. This is where OS-9 looks for executable modules on the disk. The DEFS directory holds all the files used with assembly code, like OS9DEFS, SCFDEFS, RBFDEFS and so forth. SYS contains things like ERRMSGS and PASSWORD. There also may other ones for specific devices that were sold with your particular system. If you have a tape drive, perhaps there is one called TAPE. A hard disk may have a directory called HARDDISK.

Finally, there are the ones that you create. This is where we will all have different viewpoints. It is a matter of personal choice on how to organize. Let me tell about a few of mine. For the columns that I write I have directory called COLUMN on my usual column's disk and its backup disk. For finished programs, I store them on a disk with directories named after the program. In each one I save the source code, executable module, documentation and history.

I usually keep my directories a few levels deep. If your storage media is larger, it may be useful to have a more complex directory system. I saw one system on hard disk that organized program development something like this. There were three program development areas. They were TEST, PRODUCTION, and ARCHIVE. New

developments were done in TEST. PRODUCTION held the current items. And ARCHIVE held old, more or less obsolete programs. Each of these directories had subdirectories. They were SOURCE, CMDS, and DOC. Under these were the actual files. SOURCE had another directory in it. Besides the source code files, it had LISTINGS, which held the compiler listings of the programs. This system kept track of the different type of files by using extensions like .BAS, .C, etc. But things could easily be complicated more separating files with directories like BASIC09, C_SOURCES, PASCAL and ASSEMBLY_LANGUAGE. Again, it ends up...how do you want to organize it?!!

There are a number of OS-9 commands that are are definitely needed for working with directories. The first is MAKDIR. Its syntax is simple

makdir <path>

The path can be one in the current directory. It can be a full path name. The path is passed to I$MAKDIR. It creates the new directory with bit 7 set. The new directory will contain no entries with the exception of "." and "..". Again, these are the pointers to itself and its parent directory. Once the new directory is created, it can be filled with whatever files you desire.

Two commands that are very important are CHD and CHX. In fact, these are important enough that they are built into the shell. Actually, they are there so that the execution and working directories can be changed. Usually during booting the system, these are initially set. Most systems set the execution directory to /D0/CMDS and the working directory to /D0. This simplifies things immensely. Imagine using full pathnames every time you wanted to do something. How about this?

```
/D0/CMDS/COPY /D0/DATA/MYFILE
/D1/NEWDATA/NEWFILE
```

If I some time earlier had entered:

```
CHX /D0/CMDS
CHD /D1/NEWDATA
```

then I could easily use:

```
COPY /D0/DATA/MYFILE NEWFILE
```

Copy will be executed from /D0/CMDS. The full pathname will be used for the input file and outputted to the working data directory, /D1/NEWDATA.

There are tqwo commands that go with CHX and CHD. They are PXD and PWD. They tell the you where you are at in the execution and working directory. What they do is to determine the path back as far as the device. So, entering:

PXD

might return something like:

/D0/CMDS

You may not always use them, but if you find yourself lost they can sure help.

Directories cannot be ignored when running OS-9. Being able to change directories and specify path names with them is an important aspect. I probably sound like I am preaching, but many of the problems that I am confronted with involve getting lost or not knowing where files are located. I strongly recommend anyone new to OS-9 and not familiar with hierarchical directory structures to read the OS-9 manual and learn how to work with them. And if you,are familiar with them, use them to your advantage.

THE INPUT BUFFER

This is one of those observations that you may find helpful and interesting. When you get that familiar prompt from the OS-9 shell, you can use CNTRL-A to repeat the last line typed. A CNTRL-X puts the cursor back up to the prompt. And there is backspace too. Now, anything typed goes into the input buffer. So when you want to repeat the last line typed, entering a CNTRL-A will bring it back. If an error was made, the line can be repeated. Backspace to the mistake, correct it and hit CNTRL-A to return the rest of the line. Now the corrected line can be retried.

Characters are entered into the buffer in the same order they are typed. What is more important, the buffer is not erased. Even though previous mistakes are corrected, parts of the buffer remain. Let us look at an example. Consider the following line:

```
COPY /D0/NEW_PROGRAMS/SOURCE/TEXT.C
/D1/BACKUPS/SOURCE/TEXT.C
```

I got an error #216, so I wanted to see what was in the first directory,
/D0/NEW_PROGRAMS/SOURCE. So I entered DIR with 1 space to overwrite COPY.

CNTRL-A repeated the previous line, with DIR instead of COPY. Next I held down the backspace (CNTRL-H) and backed the cursor up to end of the first SOURCE in the line. The new line looked like:

```
DIR  /D0/NEW_PROGRAMS/SOURCE
```

Entering a return, I could see the directory. It was suppose to be TEST.C. Not what I had typed. I re-entered COPY and a CNTRL-A. The line extended to where the first SOURCE ended. I typed, "/TESq" to overwrite the EOL character from the previous line and correct the mispelling. Another CNTRL-A and the entire line appeared from the earlier line. Now I backspaced to where the second mistake was and changed the X to a S. A final CNTRL-A and the line I first intended was restored with the necessary corrections. All I had to do was hit the ENTER key. If I really am rolling, I may enter a CNTRL-M, which is like hitting the ENTER.

This may seem complicated, but with a little practice, you can edit previous lines. Just remember everything is there in the input buffer. It will remain until it gets typed over. Maybe someone will come up with a new shell that has better editing feature. But in the meantime, practice using CNTRL-A, CNTRL-M, backspace and toss in a CNTRL-M from time-to-time.

```
PROMPT COPY TO SELECTIVELY COPY FILES
```

This month's program is one that I have been meaning to write for a long time. You may find it useful. I call it PCOPY. And it is written in BASIC09. It's rather simple. It opens the current working directory and reads the file entries one-by-one. As each one is encountered, it prompts for a decision whether to copy it to the output directory. A "Y" or "y" will copy it.

A few things are worthwhile to point out. I opened the current working data directory using the filename ".". The mode to open is READ+DIR. This is necessary, since it is a directory. If you work in another language. A similar method should be used. Only entries that are not dot, dot-dot or null are offered. Rather then design a copy program, I use the OS-9 command COPY, by using LOAD and later UNLINK. You may want to customize the use of it. Add -S for single drive use. Increase its memory use.

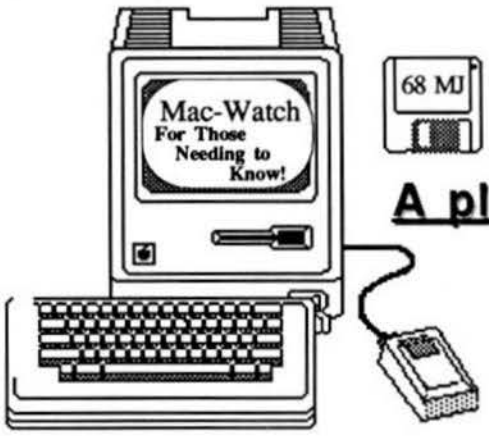
The one routine that is external is SYSTOBAS. This one translates the directory entry to a file name. Basically, it builds the name using the characters of the directory entry. Encountering a character whose MSB is set signals the end of the name. It corrects this and returns to the caller.

I believe you'll find PCOPY easy to use. If you want, make any improvements or changes. I think it can be easily rewritten into another language. This one should be easy to reproduce in assembly code. Give it shot! Until next time, have fun.

```
LISTING

PROCEDURE Pcopy
0000        (* ************************
001D        (*
0020        (* Name: PCOPY
002E        (* Author: Ron Voigts
0043        (* Date: 22-MAR-87
0055        (* Version: 1.00   Original
0070        (*
0073        (* ************************
0090        (*
0093        (* FUNCTION:
009F        (* This programs allows copying on a
00C4        (* file by file bases.  It prompts
00E6        (* for each file requiring a
             response
010B        (* from the user.  Files will be
             copied from
0138        (* the current data directory to a
015B        (* specified directory.
0172        (*
0175        (* ************************
0192        (*
0195        (* Outside modules:
01A8        (* 1. systobas
01B6        (* 2. copy ( OS-9 command )
01D1        (* 3. load ( OS-9 command )
01EC        (* 4. load ( OS-9 command )
0207        (*
020A        (* ************************
0227        (*
022A        (* Usage:
0233        (* OS9: pcopy("output directory")
0254        (*
0257        (* ************************
0274
0275        (* Output directory
0289        PARAM directory:STRING[126]
0295
0296        (* Variables
02A2        DIM path:INTEGER
02A9        DIM entry(32):BYTE
02B5        DIM name:STRING[29]
02C1        DIM answer:STRING
02C8
02C9        (* open path to current directory
02EA        (* and do other initializations
0309        SHELL "load copy"
0316        OPEN #path,".":READ+DIR
```

```
0322
0323        (* start reading the directory
0341        WHILE NOT(EOF(#path)) DO
034C          GET #path,entry
0356          IF entry(1)<>$AE AND entry(1)<>$2E
               AND entry(1)<>$00 THEN
0379            RUN systobas(entry,name)
0388            PRINT "Copy "; name; " to ";
                  directory; " ? ";
03A7            INPUT "(Y/N) ",answer
03B5            IF LEFT$(answer,1)="Y" OR
                    LEFT$(answer,1)="y" THEN
03D0              SHELL "COPY "+"./"+name+"
                  "+directory+"/"+name
03F2            ENDIF
03F4          ENDIF
03F6        ENDWHILE
03FA        SHELL "unlink copy"
0409        END
040B

PROCEDURE Systobas
0000        (* ************************
001D        (*
0020        (* Name: SYSTOBAS
0031        (* Author: Ron Voigts
0046        (* Date: 28-MAR-87
0058        (* Version: 1.00   Original
0073        (*
0076        (* ************************
0093        (*
0096        (* Function:
00A2        (* This procedure converts a 32
             directory entry
00D1        (* into a string of 29 bytes or
             less.
00F6        (*
00F9        (* ************************
0116        (*
0119        (* Usage:
0122        (* DIM entry(32):BYTE
0137        (* DIM name:STRING[29]
014D        (* RUN systobas( entry, name )
016B        (*
016E        (* ************************
018B
018C        (* Passed parameters
01A0        PARAM entry(32):BYTE
01AC        PARAM name:STRING[29]
01B8
01B9        (* Other variables used
01D0        DIM i:INTEGER
01D7        DIM fixing_it:BOOLEAN
01DE
01DF        (* Initialize the variables
01FA        name:=""
0201        i:=0
0208        fixing_it:=TRUE
020E
020F        (* Convert the entry to a name
022D        WHILE fixing_it DO
0236          i:=i+1
0241          IF entry(i)>=$80 THEN
0251            name:=name+CHR$(entry(i)-$80)
0265            fixing_it:=FALSE
026B          ELSE
026F            name:=name+CHR$(entry(i))
027F          ENDIF
0281        ENDWHILE
0285        END
```

EOF



The Macintosh™ Section

Reserved as a

A place for your thoughts

And ours.....

Mac-Watch

Yet, Another Spelling Checker & More!

It was just a few months ago I reviewed a spelling checker and commented that it was far above anything we had received or used up to that time. And it was, *but*, since then we have received several other excellent spelling checkers. Each with some very nice but not duplicated features. The one I want to tell you about this time is the one that I use for interactive checking. **MacLightning** from Target Software, Inc.

It seems, in this business, that the old adage about "*rain and pouring*" was never truer. As you know we have been doing our typesetting using a modified Macintosh computer and a laserwriter printer for 68 Micro Journal. This holding with our intention to publish 68 Micro Journal with those tools available to all of us. After all that was what it was all about back there 10 years ago, or so, when we first started publication. And we have stuck to that premise, we have always typeset every page with one of our own 68XX(X) computers.

Before Apple brought out the LaserWriter we used a high quality daisy wheel printer and various word and text processors, notably TSC Edit, Stylo in several configurations and the PAT/JUST combo. Then we switched to the modified Mac+ and LaserWriter due to the laser printer quality. Of course we were still true to our original intent of "our kind of computer, etc." as the Mac+ and the LaserWriter both are driven by a 68000 CPU.

We were an early beta test site for the "PageMaker" software system for setting whole pages for publication. Also we later received Ready Set Go to add to our set-up software. These two programs were adequate (barely so) for our needs, as was MacWrite for our text editing chores to feed our laser printer. But we made do, as that was a step forward in print quality. However, our most pressing need during all that time, and actually until just recently, was a really good spelling checker.

We had received several spelling checkers previously but they had left much to be desired...too much. So we limped along (with many more spelling errors than now) waiting and hoping for the *real thing*. Then the "pouring" started. First with the Spellswell spelling checker, reviewed here recently and then a few months later four (4) more. Two stand alone and two others embedded in (WriteNow) a word and text processor and ReadySetGo III another page make-up software program. The stand alone spelling checkers are Thunder from Batteries Included, Inc., and MacLightning, the subject of this review.

Interactive Mode

MacLightning is both a stand alone spelling checker and an interactive (check as you type) checker. It is the interactive portion that we use for most all our keyboard input activity.

While the stand alone (check a completed document or selection) mode is adequate, I find it somewhat less than Spellswell. As a result I actually use two checkers. MacLightning for interactive (while I am typing) and SpellsWell for document checking. Actually each of the others have features that would be nice to use, but the burden of trying to keep more than two dictionaries up is too much. And having told you about SpellsWell in an earlier issue, I will devote the remainded of this review to MacLightning.

MacLightning is installed into your system file as a DA (desk accessory). It has one of, if not, the most complete set of dictionaries of any spelling checker I have ever used. It comes with one standard dictionary, the Merriam Webster's Ninth New Collegiate Dictionary (93,000 words plus) and optional 45,000 word Thesaurus, 35,000 word Legal Dictionary and 28,000 word Medical Dictionary. More about these later.

While typing, Maclightning *watches* as you type. If it detects a mistake in grammar or spelling it beeps to alert you. You can then view the dictionary library by either typing a command key or select from the pull down menu.

Actual picture of this page with the dictionary library as viewed after being alerted to a typing mistake.

While in the dictionary display mode the user has several options.

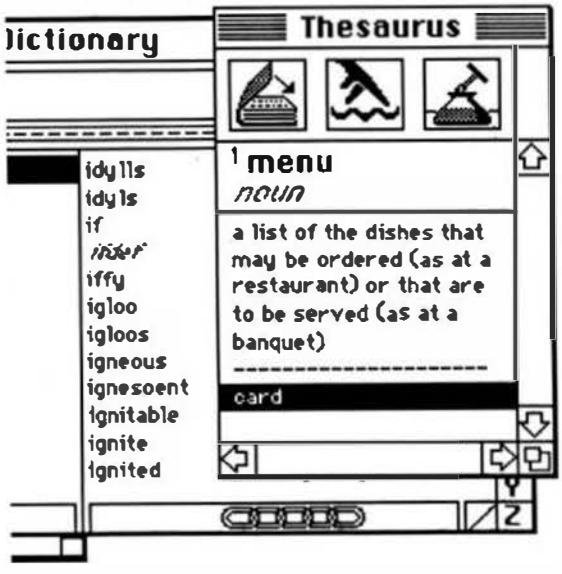
1. The word in dispute is displayed in the upper window in the library display. It may be edited directly there and then "pasted" immediately into the text file. If this is selected the word in the text file is overwritten automatically with the corrected word and you may continue your typing.

2. If you are not sure of the proper spelling you can choose to click in the "walking fingers" box of the library and MacLightning will attempt to find the correct spelling. Due to the large size of the dictionary, it's most likely you will only have to click on the found correct spelling and it will replace the offending incorrect word. This feature also allows you to select the library and then type in what you believe to be the correct spelling of any word. Then by clicking the "walking finger" you can look up any word in the dictionary. Once found it can be placed in the text file, as outlined above. This is a handy feature and can even be used to look up a word from outside any application. just activate the speller from the Apple DA Menu and type in the first few letters of any word and proceed as above. Only thing is, there will be no text file to correct. This way it allows you to use it as an electronic dictionary, any time.

3. Once you have found the correct spelling for your word in question there are several other functions that makes this a first class package.

The Thesaurus containing over 45,000 complete words including suffixes and prefixes. This allows a combination of over 1.4 million possible combinations of synonyms. A definition of each word is also displayed. This display may even be selected and then pasted directly to your document.

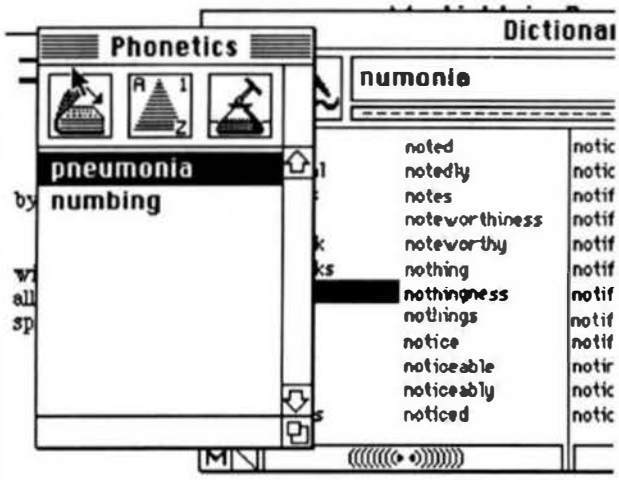
This allows virtual cross referencing and you may select and replace your original word with any of the displayed synonyms.



The Thesaurus selected

As you can see, the word menu could be replaced by the noun "card".

Another function is a "phonetic" or "sound alike" window. This window is a standard feature and allows the display of words that sound alike but are spelled differently.



The Phonetics window - "numonia" was selected - the proper spelling but sound alike is located and displayed selected. This may be then clicked directly into your text file.

These features make MacLightning a very powerful interactive spelling checker.

Should you opt for the additional checkers, Legal and/or Medical, the additional dictionaries would be also on line at all times.

This is test setion that we wil use to display the displays ioin the "selection" mode. Bear in mind that this is intenionally typed with errors for demonstration. Also it is very short. MacLightning will check about 75 or more words per second, so larger selections or documents are no porblem, time wise.

In this mode you are informed of how many words were checked, how many were suspect, their average length and how long the longest word was. In the Misspelled word window are all of the suspect words. These will be initially displayed in found order. By clicking the "A - Z" middle box, they will be sorted and all duplications reduced to one each of that word. Each one may be selected and pasted into the library window, there it can be managed as any of the above explained functions.

As mentioned earlier, most all of the functions have menu and command key equivalents.

The one feature that I dislike about this mode, and the primary reason I do not use it for selection or document checking is that in the replace function, in this mode, it uses the host applications *find and replace* function. Compared to most all the others it is slow and cumbersome. If they ever get that changed to work as smooth as the other portions of MacLightning, I suspect that it will end up being our one and only.

Words may be entered into the dictionary at any time the library window is open. Words the user enters in are displayed as italics. They may also be removed at any time. However, the words that were in the original dictionary may never be removed. They are locked in. This feature can save a lot grief for those that might click on the wrong box. Several other checkers we have tested do not have this fail-safe feature, and disaster lurks therein.

MacLightning also does a nice job of grammar checking. It checks and alerts if you start a sentence with a lower case character. It checks for repeated words (to to, etc.). It can catch most apostrophes incorrectly placed in a contraction. Also it catches capital characters placed in the body of a whole word.

MacLightning works well with 512K and greater Macs. It will not run on the 128K Mac or the Mac XL.

Text may be loaded to the clipboard and checked by MacLightning as well as checking text only files. It is compatible with PageMaker 1.2, ReadySetGo (version three has checker built in) Word and about 90% of all the other Macintosh software. With Multiplan and Excel it runs in the interactive mode only.

From:

Target Software
14206 Southwest 136th Street
Miami, Fl 33186

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

FORTH AND THE 6820/6821 PIA

I received a telephone call the other day from a reader who was having trouble getting his FORTH to interface properly with a PIA (Parallel Interface Adapter), so, since I have also had that sort of trouble, I thought that this would be a good time to review some of the techniques.

Actually, programming the PIA is relatively easy, once you know what you want to accomplish, but some of the programming instructions I have seen have been pretty opaque.

6820/6821 Characteristics

Whenever the PIA is reset, all 16 normal I/O lines, plus the two lines, CA2 and CB2, go into an indeterminate state, which looks like a TTL high to the outside world. CA1 and CB1 are input, only, so we don't have to worry about their affect on the outside world. However, in any control operation where one or more of the I/O lines is connected to a level-sensitive input device which is turned on before the computer, we have to assume that a high at that device's input is the default condition. Therefore, when we initialize the PIA, we want to be sure that signal logic does not change at the moment of PIA initialization, since that might have an effect on the external device.

There is a potential "gotcha" with the 6820/6821 in that all eight lines on the B side are tristate, but all eight lines on the A side are not, so that there can be considerable line loading by any inactive lines on the A side. You may not find this to be a problem, but you have been warned!

One of the problems with the 6820/6821 is that there are three registers, but only two allowed control line states. Therefore, you must select one of the registers by first selecting and setting a different register. You can program the Data Direction Register (DDR), also known at the Peripheral Interface Register, only by first writing a 0 into bit #2 of the Control Register (CR). The CR and the Peripheral Data Register (PDR) are accessible directly from the control lines, RS0 and RS1, which are normally connected to the address lines A0 and A1.

CONTROL REGISTER PROGRAMMING

Programming the control register is the one which causes the confusion. Part of the problem is that the A and B sides, naturally, are not identical in their operation as input lines. However, during programming, there is no difference between CAn and CBn. In any case, if you do not have any specific plans to use one of the control register lines, no matter which of the four it might be, it should be specifically masked or set as an output in order to prevent accidental interrupts because of stray electrical noise. I learned this particular tip by bitter experience in an industrial lab application.

Figure 1 shows the bits which must be programed in the CR. Bit #6 and bit #7 are always set to 0 during programming, though they indicate the presence of an interrupt during operation of the PIA.

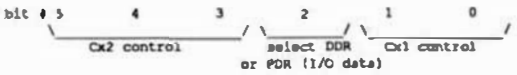


Figure 1. The programmable bits of a Control Register (x represents either A or B).

Figure 2 shows all of the steps, in the required sequence, for properly programming the PIA. Of course, some of the steps are combined into multiple-purpose commands, so that you don't actually do all of the steps, individually; the machine does some of them for you. Steps 1, 2, and 3 are one command; steps 4 and 5 are one command; and steps 6, 7, and 8 are one command. This can be seen in the listings for screen #17-18 and screen #26, which show two versions of PIA initialization. Line# 1 of screen #17 is a good example of the combination of steps 1, 2, and 3.

(1) Reset the PIA.
(2) Select the Control Register.
(3) Write a 0 into bit #2 of the Control Register to enable selection of the Data Direction Register.
(4) Select the Data Direction Register.
(5) Program the Data Direction Register for the appropriate input and output lines.
(6) Select the Control Register.
(7) Program the Control Register for the appropriate interrupt and I/O operation.
(8) Select the Peripheral Data Register.

Figure 2. All of the sequential steps in programming the PIA.

Setting bit #0, #1, #3, #4, and #5 all to 0 has the effect of a RESET on the PIA. Setting RS0 to 0 and RS1 to 1, via the addressing lines, selects the Control Register. Setting bit #2 to 0 permits selecting the DDR.

The two lines, CA2 and CB2, can be used as control or strobe lines by properly programming bit #3, #4, and #5 of the CR. This is shown in Figure 3.

bit #: 5	4	3	interrupt output
Ca2 input			
‾|_ 0	0	1	‾|_ (interrupt)
_|‾ 0	1	1	‾|_ (interrupt)

			Cx2 output
1	0	0	‾|_|‾ (handshake1)
1	0	1	‾|_|‾ (handshake2)
1	1	0	0 (holds low)
1	1	1	1 (holds high)

Figure 3. The only valid programming codes for Cx2.

Notice that Bit #5 is always low for an input (interrupt process) and always high for an output (control process). Furthermore, there are only six useful programming combinations out of the eight possibilities.

Once an interrupt has been recognized, then bit #6 of each control register can be examined. The interrupt request through Cx2 will have caused bit #6 to be set high, and it will be reset to low by a reading of the PDR. The polling of bit #6 before reading the data will indicate which half of which PIA caused the interrupt.

The two lines, CA1 and CB1, can only be used for accepting interrupts, as shown in Figure 4.

bit6: 1	0	interrupt output
Cx1 input		
‾|_ 0	1	‾|_ (interrupt)
_|‾ 1	1	‾|_ (interrupt)

Figure 4. The only valid programming codes for Cx1.

Bit #7 is set high by a transition of Cx1 and cleared by reading the PDR. The polling of bit #7 before reading the data will indicate which half of which PIA caused the interrupt.

I really don't want to say very much at this time about interrupts and FORTH, since I don't have enough personal experience with that combination at this time. Maybe I will have a chance to do some experiments so that I can report later. Also, I will save the topic of handshaking to another time.

setting	result
0	selects the Data Direction Register
1	selects the Peripheral Data Register

Figure 5. Effect of programming Bit #2 of the Control Register

The only item left for discussion is bit #2. This bit produces the results shown in Figure 5.

Data Direction Register Programming

There is very little which needs to be said about programming the DDR, except that once a selection of input or output direction has been programed, you cannot change it without going through another initialization routine. Keep in mind that the 6820/6821 PIA was not designed so that you could get both input and output on the same line without going through some intermediate programming. Programming a line high makes it an output line, and programming a line low makes it an input line.

Some Practical Experiments With FORTH

My experiments reported here were done with a modified SWTPC MP- LA parallel interface, which used the venerable 6820 (now improved to the 6821). My modifications involved jumpering around the DM8833 bi-directional transcievers. I did this so that I could have full software control of the data direction for each bit. The MP-LA, as originally designed, forced you to select the data direction for each bit in fixed groups of four. Therefore, you could not have three input and five output bits on side A, and seven input bits and one output bit on side B, or any other such combination. Furthermore, the data direction was hard-wired, so that you could only change direction with a soldering iron.

As a further problem, the MP-LA was configured for a 6800, not a 6809. The address sequence was set to A-side I/O, A-side control register, B-side I/O, and B-side control register. For the 6809, a much more favorable arrangement would have been A-side I/O, B- side I/O, A-side control register, B-side control register. You can make this change on the MP-LA, or similar board, if you are handy with a knife and a soldering iron. You simply reverse the wiring to the RS0 and RS1 addressing lines, as show in Figure 6.

	RS0	RS1
Original SWTPC addressing	A0	A1
New addressing scheme	A1	A0

Figure 6. Improved addressing scheme for the PIA.

The effect on the apparent addressing of the various PIA registers is shown in Figure 7.

	RS0	RS1	apparent equivalent address
	A0	A1	
A-side I/O	low	low	xxx0
A-side control register	high	low	xxx1
B-side I/O	low	high	xxx2
B-side control register	high	high	xxx3
	A1	A0	
A-side I/O	low	low	xxx0
B-side I/O	high	low	xxx1
A-side control register	low	high	xxx2
B-side control register	high	high	xxx3

Figure 7. Results of the changed PIA addressing scheme.

Actually, I use the wiring shown schematically in Figure 8. I have a DPDT slide switch hot-melt glued to the MP-LA circuit board and wired so that one position of the switch produces "normal" addressing and the other position of the switch produces the more efficient addressing. That way, I can change to accommodate changing software.

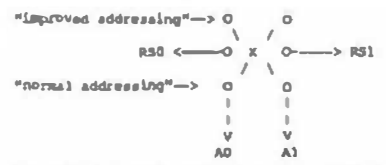


Figure 8. Wiring diagram for a DPDT switch for changing the address lines on a MP-LA circuit board.

I recommend doing it on an individual board, rather than to the mother board, since you would then make your total system imcompatible with commercial software. I can't take credit for this idea (except for the switch), since I have seen it before, but I can't remember where. It is unfortunate that SWTPC did not use it, since I don't think it would have done any harm to the ACIA programming, either.

The advantage of this change is that it will let you address the PIA as either an 8-bit or a 16-bit interface. If you use the D register commands, you can shave cycles off getting the same 16-bit information with separate A and B commands. The saving will be either 3 cycles for indexed or 4 cycles for extended addressing. This can mount up to a significant saving over a long sequence of data movement. One pair of examples shows a 12% improvement.

You still use this addressing scheme to advantage, even if you are using a 6800, etc. Certainly, it would be an advantage for the 68xxx. FORTH is inherently a 16-bit language, so the differences among the 6800, the 6809, and the 68xxx processors can disappear when addressing the 6820/6821.

PROGRAMMING THE MP-LA

I assumed that most of you would have the "standard" PIA addressing, so I wrote a set of definitions with that in mind. I will detail just how I arrived at the constants I used in my experiments.

HOUSEKEEPING

The first thing that I did was to define four constants and one variable, since I wanted to write a very general set of definitions which I could put into my FORTH utilities "tool box". If there is anything I hate, it is having to write a nuisance definition more than once!

I/O PROGRAMMING

I made an experimental setup with LEDs to indicate the state of each of the 16 conventional I/O lines and CA2 and CB2. It follows, then, that I wanted to initialize with all I/O lines as output lines, with the level set high to turn off the LEDs. Since all 8 bits high is the decimal number 255, that has been defined as a CONSTANT in order to speed up the PIA initialization.

The two constants named 54 and 62 are used to strobe CA2 and CB2 low (54) and then back high (62). Again, we want to use these values as CONSTANTs, since we want the strobing to be as fast as the machine will allow, and not limited by unnecessary software limitations.

```
bit#:    5   4   3   |   2   |   1   0
setting: 1   1   0   |   1   |   1   0
         Cx2 always low |  PDR  | no interrupts
```

Figure 9. Rational for 54 as a programming constant.

```
bit#:    5   4   3   |   2   |   1   0
setting: 1   1   1   |   1   |   1   0
         Cx2 always high |  PDR  | no interrupts
```

Figure 10. Rational for 62 as a programming constant.

At this stage, we are not considering whether or not to accept interrupts through CA1 or CA2, so the 54 and 62 mask off their interrupt capability. Refer to Figures 9 and 10 for the details.

The 255 was made a CONSTANT because it is used to set all of the I/O lines as output.

Also, in order that I would not have to keep remembering the base address of the PIA, I set it to a CONSTANT named (after a lot of deep thought) PORT.

Since it is much easier to load the contents of a VARIABLE than it is to do a lot of stack convolutions, I created the variable named _PORT. You may wonder why I used the _ prefix: I now use this prefix with all of my variables, so that I have the extra built-in documentation. The _ represents a blank to be filled in.

The definitions shown in screen #17-18 and screen #25 accomplish the same thing in initializing the PIA. The difference is that the longer definition, INITIALIZE-PIA , in screen #17-18 works on the "standard" addressing scheme, while INITIALIZE-PIA1 in screen #25 works on the "improved" addressing scheme.

Screen #17 is split into four words to be used for independently initializing each of the four possible sections of the PIA. This way I could change one of the B-side lines from input to output, or vice versa, without affecting the A-side lines.

There is a problem with initializing A and B, since the definitions, as shown, will also de-program Cx1 and Cx2. So you will either have to rewrite the definitions I wrote or reinitialize the associated control lines, if you are using them. As it stands, the definitions (INITIALIZE-PIA-A) and (INITIALIZE- PIA-B) leave the control lines in an indeterminate state.

Several utility words are defined in screen #19-20, and don't need any further explanation.

Execution Speed

In any discussion of addressing peripherals, the question of execution speed always comes up, as it should. Therefore, I wrote several test programs to determine the effect of various programming changes on the running time for PIA access. These test programs are shown in screen #21-24 and 26. The results are tabulated in Figure 11.

	condition tested	number of iterations	total sec.	1 pass ms.	net ms.
TEST1	empty loop	50,000	2.00	.040	.000
TEST2	typical FORTH	50,000	23.80	.476	.436
TEST4	faster FORTH	50,000	11.40	.228	.188
TEST3	fast machine lang.	50,000	3.50	.070	.030
EXERCISE-PIA		65,535	17.00		
EXERCISE-PIA1		65,535	15.76		

Figure 11. Results of some speed tests with FORTH on a 6809 running at 1 MHz.

The first and last instruction in each test listing are commands to CB2 to control an external clock used for timing the tests.

Each of the first four tests were run 10 times and the last two were run 5 times, to get the average running time reported under "total sec." The next column shows the time in milliseconds for a single pass through the loop, and the last column shows

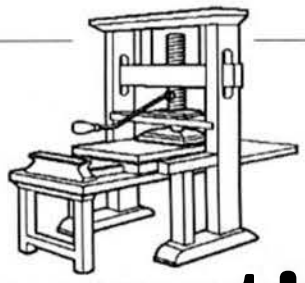
the time required for one pass after subtracting the loop overhead.

The test set shows that the worst case for FORTH was only about one order of magnitude worse than the fast machine language version. Granted that pure Assembly language would be slightly faster, but how many situations do you know of where a pulsing rate of under 1 ms. would be too slow! Of course, these tests leave no time for doing anything else, but that is typical of any benchmark.

The significance of the last two tests is in illustrating the increased efficiency of making use of the D register to send data to the PIA, as compared to the conventional way. The improvement is about 12%, which is very gratifying, particularly when you consider the easier programming.

```
SCR # 16
  0 \ CONSTANTS                                          04/05/87
  1
  2    54  CONSTANT   54
  3    62  CONSTANT   62
  4   255  CONSTANT  255
  5 57456  CONSTANT  PORT
  6
  7        VARIABLE  _PORT

SCR # 17
  0 : (INITIALIZE-PIA-A)     ( n -- )                    \ RDL 04/06/87
  1     0    _PORT  @  1+    C!        \ equivalent to RESET A
  2     ( n )  _PORT  @    C!          \ set A-side DDR
  3     4    _PORT  @  1+    C! ;      \ enable A-side PDR
  4
  5 : (INITIALIZE-PIA-B)     ( n -- )                    \ RDL 04/06/87
  6     0    _PORT  @  2  +    C!      \ equivalent to RESET B
  7     ( n )    _PORT  @  2+    C!    \ set B-side DDR
  8     4    _PORT  @  2  +    C! ;    \ enable B-side PDR
  9
 10 : (INITIALIZE-PIA-CA)   ( n -- )                    \ RDL 04/06/87
 11     ( n )    _PORT  @  1+    C! ; \ set CA1 & CA2
 12
 13 : (INITIALIZE-PIA-CB)   ( n -- )                    \ RDL 04/06/87
 14     ( n )    _PORT  @  2  +  C! ; \ set CB1 & CB2

SCR # 18
  0 : (INITIALIZE-PIA) ( nA nB nCA nCB -- )             \ RDL 04/06/87
  1     >R  >R  >R                  \ temporarily store nCB, nCA, nB
  2     ( nA )  (INITIALIZE-PIA-A)
  3     R>     (INITIALIZE-PIA-B)
  4     R>     (INITIALIZE-PIA-CA)
  5     R>     (INITIALIZE-PIA-CB) ;
  6
  7 : INITIALIZE-PIA    ( nA nB nCA nCB adr -- )        \ RDL 04/06/87
  8     ( adr )  _PORT  !
  9     (INITIALIZE-PIA) ;
 10
 11 \ nA = DDR code for A-side.
 12 \ nB = DDR code for B-side.
 13 \ nCA = CR code for A-side.
 14 \ nCB = CR code for B-side.
 15 \ adr = PORT first address.

SCR # 19
  0 \ STROBE-PIA-LOW  STROBE-A-LOW  STROBE-B-LOW          04/05/87
  1
  2 : STROBE-PIA-LOW    ( adr -- )                      \ RDL 04/05/87
  3     ( adr )  DUP              \ assume CA2 is presently high
  4     54     SWAP     C!        \ bring CA2 low
  5     62     SWAP     C! ;      \ return CA2 to high
  6
  7 : STROBE-A-LOW      ( -- )                          \ RDL 04/05/87
  8     _PORT  @  1+
  9     STROBE-PIA-LOW ;
 10
 11 : STROBE-B-LOW      ( -- )                          \ RDL 04/05/87
 12     _PORT  @  3  +
 13     STROBE-PIA-LOW ;

SCR # 20
  0 \ OUT-PIA-A  OUT-PIA-B  IN-PIA-A  IN-PIA-B           04/05/87
  1
  2 : OUT-PIA-A    ( b -- )                             \ RDL 04/05/87
  3     ( b )  _PORT @  C! ;
  4
  5 : OUT-PIA-B    ( b -- )                             \ RDL 04/05/87
  6     ( b )  _PORT @  2+  C! ;
  7
  8 : IN-PIA-A     ( -- b )                             \ RDL 04/05/87
  9     _PORT @  C@ ;
 10
 11 : IN-PIA-B     ( -- b )                             \ RDL 04/05/87
 12     _PORT @  2+  C@ ;

SCR # 21
  0 : EXERCISE-PIA      ( adr -- )                      \ RDL 04/05/87



  1     62  _PORT  @  3  +  C!              \ bring CB2 high
  2     255  _PORT  @  C!    255  _PORT  @  2+  C!
  3     256  0  DO
  4       255    1    -  _PORT  @  C!
  5       256  0  DO
  6         255    1    -  _PORT  @  2+  C!
  7       LOOP
  8     LOOP
  9     54  _PORT  @  3  +  C! ;             \ bring CB2 low
 10
 11 \ Average of 5 runs = 17.80 sec

SCR # 22
  0 \ TEST1  TEST2                                       04/05/87
  1
  2 : TEST1  ( -- )                                      \ RDL 04/05/87
  3     62  _PORT  @  3  +  C!              \ bring CB2 high
  4     50000  0  DO
  5     LOOP
  6     54  _PORT  @  3  +  C! ;            \ bring CB2 low
  7 \ Average of 10 tests = 2.00 sec, .040 msec/loop
  8
  9 : TEST2  ( -- )                                      \ RDL 04/05/87
 10     62  _PORT  @  3  +  C!              \ bring CB2 high
 11     50000  0  DO
 12       STROBE-A
 13     LOOP
 14     54  _PORT  @  3  +  C! ;            \ bring CB2 low
 15 \ Average of 10 tests = 25.80 sec, .476 msec/loop

SCR # 23
  0 \ STROBE-A-ML  TEST3                                 04/05/87
  1
  2 CODE  STROBE-A-ML  ( -- )                            \ RDL 04/05/87
  3     15926  #       LDD
  4     PORT  1  +     STA
  5     PORT  1  +     STB
  6     NEXT,
  7 END-CODE
  8
  9 : TEST3  ( -- )                                      \ RDL 04/05/87
 10     62  _PORT  @  3  +  C!              \ bring CB2 high
 11     50000  0  DO
 12       STROBE-A-ML
 13     LOOP
 14     54  _PORT  @  3  +  C! ;            \ bring CB2 low
 15 \ Average of 10 tests = 3.50 sec, .070 msec/loop

SCR # 24
  0 \ PORT+1  TEST4                                      04/05/87
  1
  2 PORT  1+  CONSTANT  PORT+1
  3
  4 : TEST4  ( -- )                                      \ RDL 04/05/87
  5     62  _PORT  @  3  +  C!              \ bring CB2 high
  6     50000  0  DO
  7       62  PORT+1  C!
  8       54  PORT+1  C!
  9     LOOP
 10     54  _PORT  @  3  +  C! ;            \ bring CB2 low
 11
 12 \ Average of 10 tests = 11.40 sec, .228 msec/loop

SCR # 25
  0 : INITIALIZE-PIA1   ( adr -- )                       \ RDL 04/05/87
  1     ( adr )  _PORT  !
  2     0    _PORT  @  2+    !        \ equivalent to RESET
  3     65535    _PORT  @    !        \ set all lines to output
  4     15934    _PORT  @  2+    !    \ set CA2 and CB2 to high
  5     65535    _PORT  @    ! ;      \ set all 16 lines to high

SCR # 26
  0 : EXERCISE-PIA1     ( adr -- )                       \ RDL 04/05/87
  1     62  _PORT  @  3  +  C!              \ bring CB2 high
  2     65535  _PORT  @  !
  3     65535  0  DO
  4       65535    I    -    _PORT  @    !
  5     LOOP
  6     54  _PORT  @  3  +  C! ;            \ bring CB2 low
  7
  8 \ Average of 5 runs = 15.76 sec

SCR
```

The "Grandfather of Desk Top Publishing"



The Editor's Place to Comment!

Ramblings & What-not...

This months Ramblings are just that. No particular subject or order, just several things I thought you might be interested in. Some biased and some I'll let fall as I received them. Also some other things of general interest.

First off, some books received

68000 Microcomputer Systems Designing & Troubleshooting

Prentice-Hall Hard Bound
By: Dr. Alan D. Wilcox

If you are into the 68000 from either or both software and hardware designing and implementation, then this is one of "the" tutorial books you would place on your *need-it* list. Starting at the basic planning stage it progresses for 570 pages of solid and practical applications material.

This book describes the Motorola 68000 and methods of designing a 16 bit computer using it as the main CPU. It covers very thoroughly principles and techniques that can be applied to the design, construction and troubleshooting of a 68000 project. Both from the hardware and software perspective. While recommended as a college text tutorial, I found it excellent as a reference and refresher book.

Starting with an introduction to accepted and proven engineering principles it leads to a solid foundation for a practical, working system completely debugged and documented. In the earlier chapters emphasis is centered on project planning to achieve a realistic completion schedule. Additional emphasis is directed towards systematic design techniques using the 68000 microprocessor. Many circuit examples are given and each is supported with ample and clear drawings and circuit diagrams.

As a starting point the author recommends the student begin troubleshooting experience on the Motorola 68KEBC (Educational Computer Board), and then progress to the design and construction of a 68000 SBC. For those not having an EBC, the treatment of the steps involved is sufficiently complete to not require the ECB if you decide to embark on this as a do-it-yourself project. The basic tools needed are a logic probe, oscilloscope (40 Mhz or faster) and VOM. Of course having access to a logic analyzer and other development tools will make thing easier.

Modular hardware and software design is stressed. The result is a design both flexible and adaptable as well as practical in real world values. Complete coverage is afforded system timing, worst case considerations, proper components and designing to avoid testing and manufacturing difficulties. All in all, I consider this among the better design treatments for the 68000 in my library.

Simple enough for the beginner it has a wealth of examples and suggestions that will stand well with the seasoned professional engineer, from both the hardware and software perspective.

CPM 68K is the major operating system discussed (one part line mention of OS-9 & their address & telephone number in a footnote). Other than that no serious mention is made of the several other (and probably more popular) operating systems currently running with the 68XXX series of devices. While simplicity was apparently a part of the design intent of this documentation, it is felt by this reader, possibly from a biased viewpoint, that some of the others at least deserved some exposure. Only if for the sake of completion. And my bias may be showing, but I felt that OS-9 should have been afforded additional mention. The S100 bus

(IEEE Std-696) was chosen as the backplane system. While, considering the state of things today, the S50 bus may not have been the answer, however, I would have been more appreciative had the VME or some of the other simple and/or popular configurations been included occasionally. Again, for completeness. However, as I said earlier, I heartily recommend it for what it is, and not for what it is not. It's an excellent presentation of a complex subject in a very understandable way. *Buy it, little additional effort need be expanded to adapt the principles outlined in this book to your favorite bus structure or operating system.*

- - -

Electronic Design of Microprocessor-based instruments and control systems

By: Abund Ottokar Wist & Z. H. Meiksin

Prentice-Hall Hard Bound

This book is authored, in part, by one of our "long time" subscribers, Abund Wist. Dr. Wist is and has been an S50 bus users for many years. His expertise in the microcomputer field is reflected in this guide for incorporating microprocessors into new and existing systems.

While somewhat dated due to the major examples being older microprocessors (6800, 6809, 6502, 8080, Z80, RCA COSMAC, SC/MP, etc.) it is never-the-less a valuable reference source because of the way the authors have made the complex seem so simple. Also I lay little importance to the non-use of those*older* processor today. We have yet to extract all the potential of those venerable devices.

We are daily deluged with press hype about all the new and glorious devices arriving in a seemingly endless stream on the micro scene. *I am the more reminded that we are slaves to glitter and gloss! I know of many projects that could do their intended functions well with one of those older devices, rather than the newest, biggest, and much more expensive "(ZAPPO-TR(illion)" 64 Megabit CPU. Sometimes even do it better!* And so the function of this book is not diminished to any great degree. This also is a *"must"* reader regardless of what your skill level might be.

Here are specific examples of how to increase the efficiency, speed and accuracy of monitoring and control equipment. Examples include temperature controls, scientific instruments, lab experiments and much, much more.

Practical information is the backbone of this book, the capabilities and parameters of some of the more popular 4 and 8 bit microprocessors (with some reference to 16 bitters included) and guides to which would best suit your particular application. How best to connect and test the chosen device. How to apply microcomputers to heating and cooling systems - for fire, smoke and intruder detection - an infusion system that delivers a liquid at a prescribed rate - for monitoring lab tests - for control applications in automobiles, and lots more. All explained and accompanied with graphics and diagrams to give understanding even to the less progressive of us.

Detailed examples of the basic components of a computer. How to use them and how they operate. Thorough discussions on the efficient manner of programming these applications. Evaluation of storage devices and peripherals and other devices external to the CPU. Even an analysis of the best uses of the more popular languages.

This book I recommend to those wanting to expand their level of computer knowledge. Not just because it was written by "one of us", *but because it is just as applicable today as it was the day it was published.*

A Little More on the CoCo III

I don't want to be accused of harping on a subject some hold sacred. I still believe the CoCo III is a bargain, but only if you purchase OS-9 Level II as the main operating system. OS-9 is the only thing that could have (or maybe still can) save it from a fate other failing Tandy computers have experienced.

In a recent Wall Street Journal article it was stated that Tandy had reported the only profitable computer products were their MS-DOS clones. The CoCo was reported as being down better than 38 percent in sales. Mr. Ed Juge, director of Tandy Market Planning, was quoted as saying, "It seems as if there's a big shift across the board towards the IBM compatibles." For the CoCo III and the Tandy model 4D, it is a dire condition.

I have strong feelings on this subject. However, I can't do as several suggested--"don't mention the problems or say anything that might discourage others from buying one." As if somehow that will make everything o.k.

I owe an obligation to my readers first! If I perceive or hear of a market situation that could depreciate an intended investment on their part, I have a duty to report that information. That's what we have always promised and tried to deliver. And that's what we are paid for. We

have a trust to uphold and it has been made known to all since our inception, reader and advertiser alike. The only reason we have survived so long as a magazine (now 10 years +) is that I have always tried to report in the best interest of my readers first, advertisers second and all others next. Our readers believe the claims of advertising in 68 Micro Journal (we screen claims more tightly than any other magazine I am aware of). *Without readers, we would have no advertisers.* And while we have all of you as readers, we have a solid market for those vendors who wish to advertise to those of you who may be interested in the products they have to sell. Millions of dollars worth of products are sold annually through the advertising contained in 68 Micro Journal.

So from time to time I will let you know. And for any of you who feel I am unfair or too harsh, please answer this, Have you directed as much toward the folks who made and sell it? Ask them right out, "are they attempting to sell a dying product, or are they going to stand behind it so that those who spent their hard earned dollars with them will get their moneys' worth?" Remember the Radio Shack MC-10?

Your input can go a long way in helping determine their future actions concerning the CoCo III. Mine is but a single voice. Yours and other owner/users *voiced concerns* just may be heard. I certainly hope so. Because *sticking our heads in the sand* certainly won't avail any of us very much.

All this because of two negative letters concerning my previous remarks along those lines. However, even two is enough to stir me to reply. I owe those two readers that much, That's what they paid us for when they subscribed. Agree, or disagree, it is all the same. Only when we can openly express our opinions, are we really being honest with one another.--*"That is what it is all really about."*

Gathering

Unless something changes soon, there will be no get together this summer, as I had suggested a few months back. We have received letters and telephone calls from several hundred expressing interest in such a meeting. However, it seems that there would be few vendors to display the latest products. Without them, it would not be a fair showing. After all, it was for them I had originally planned such an event. I get so much mail and telephone request for information on products advertised in 68 Micro Journal I thought that it might be a good thing. I still do! But without the "whole gang" participating it would be a two or three team tournament. So I have placed it on hold, for now. If the spark lights up I will announce it in time for all who expressed interest to be advised. And to all who cared enough to let me know, my thanks.

P.S. Let me throw a plug in here for the Data-Comp Mustang-08A. Don't overlook this system. *It has been improved considerably, without a price increase.* It is still on sale for less than two thousands dollars complete now with 4 serial ports, two parallel ports, a 12 Mhz CPU, built in battery backup and your choice of Sk*DOS or OS-9 level 2-Professional version including the $500.00 C compiler! Nowhere, is there another system like that!

Tom, from Data-Comp asked me to remind you to remember - *this low price won't last forever.* They don't have the slightest idea how long the parts price breaks will continue. When the special deal on disk controllers, disk drives (floppy and hard disk) and CPU boards is over, the price goes up nearly a thousand dollars. If ever you thought about a 68XXX system, give this system serious consideration. This may just be the best offer you will ever get! While not as powerful as the 68020 systems from GMX, Windrush Micro Systems and Data-Comp, it is still the best bargain around for a genuine complete 68XXX system! If you find a better deal you sure better snap it up, the hardware parts alone normally cost much more than the price of the complete Mustang-08A system, with software. It took a lot of buying power to get the parts for this system down to where they are now. And those things change over-night. Just thought you might like to know.

DMW

The LAB6809

An Economy 6809 Prototyping System

As most of you know, for years I have been "urging" someone out there to come up with a low cost, effective teaching and prototyping board or system. Well, in the past few months we received what I consider several excellent products along those lines for review. For this review we are looking at the Barton Laboratories **"LAB 6809"**.

This item is a standard S50 size 50 pin card, composed of the following major components and/or development provisions:

Prototyping Area
68B09 CPU
Jumper selection - 1 or 2 Mhz
on-board voltage regulation
4 - 2K RAM/ROM/EPROM sockets
MRDY cycle stretch w/synchronous clock
EIA RS-232 serial interface
4-bit latch extending addressing up to 1 Mbytes
S50 bus gold plated connectors
PICOBUG monitor - 17 commands - 24 I/O routines
Silkscreening on both sides.
Provisions for 7805, 78xx & 79xx regulators.
22 square inches of wirewrapping area.
5 select lines on 16 byte boundaries, similar to S50 slots.
2nd RS-232C port or 6840 timer/counter.
6821 PIA.

The board is constructed of G10 glass PCB, 0.062 thick copper each side. Solder masked both sides.

Power requirements:

V	Max current
+8	1 A
+16	30 mA
-16	30 mA

As any good development system should, there is a wealth of flexibility afforded by the jumper blocks provided on the board. The documentation contains complete jumper block information including tables and layouts. In fact the documentation is more than adequate. It includes a large, fold out schematic, parts list, parts layout, functional block diagram and completely commented source listing of the PICOBUG monitor.

The how-to portion of the documentation is included in the *Hardware Overview* portion of the manual. The manual is complete and the detailed discussions concerning the board are very well done. I find that an important consideration.

I thought the package well thought out and documented. I only wish we had had something of this quality and functionality back in those days when I spent lots of time wire wrapping and all that other "fun" stuff. If we had had the LAB 6809 then, things would have been a lot simpler and, of course, we wouldn't have been *reinventing the wheel* all the time!

The *theory of operation* section begins with a discussion of the voltage regulation as pertains to this particular board. Each *theory* section is additionally accompanied with matching portions of the schematic for clarity and completeness of the functional discussion.

The remainder of the theory discussion covers the following main topics:

a. Oscillator circuits
b. 6809 CPU Inputs
c. 6809 CPU Outputs
d. Decoding and memory mapping
e. Data bus control
f. Memory Ready circuits
g. Synchronization of clock and other signals
h. 6850 ACIA
i. RS-232C serial port
j. Baud rates
k. S50 Bus interfacing
l. Control Inputs
m. Constructing a power

Continued on page 37

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs. All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.

FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.

Like compilers for other languages, FORTHBUILDER can operate in "batch mode".

The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.

FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.

F, CCF, S - $99.95

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems

FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriente which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") $249.95

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.
F, S, CCF - $99.95

Macro Assembler for TSC -- The FLEX, SK*DOS STANDARD Assembler.
Special -- CCF $35.00; F, S $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF, GeneOrate OS-9 Memory modules under FLEX, SK*DOS.
*FLEX, SK*DOS, CCF, OS-9 $99.00*

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.
F, S, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.
F, S, CCF - $75.00

XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8
F, S, CCF - $98.00

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.
U - $299.99

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatterdeveloped by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Grafirax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.
* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file,
*JUST2.TXT PL9 source:FLEX, SK*DOS - CC*
Disk #2: JUSTSC object and source in C:
*FLEX, SK*DOS - OS9 - CC*
The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!
Disk (1) - PL9 FLEX only- F, S & CCF - $49.95
Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95
PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.
*Regular FLEX, SK*DOS $129.50*
** SPECIAL INTRODUCTION OFFER * $79.95*
SPECIAL PAT/JUST COMBO (w/source)
*FLEX, SK*DOS $99.95*
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00
Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.
*FLEX, SK*DOS $69.95*
BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entires lines, etc. Complete with over 25 different CRT terminal configuration overlays.
*FLEX, CCF, SK*DOS $39.95*
SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!
*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 - $175.00*
SPELLB "Computer Dictionary" from S.E. Media -- OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPLL.CMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.
F, S and CCF - $129.95
STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES 6809 CCF and CCO - $99.95,
F, S or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES 6809 CCF and CCO - $69.95,
F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES 6809 CCF and CCO - $59.95,
F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!!
F, S or O - $329.95, U - $549.95
O, 68000 $695.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DYNACALC -- Electronic Spread Sheet for the 6809 and 68000.
F, S, OS-9 and SPECIAL CCF - $200.00, U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.
F, S - $59.95, U - $89.95

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
*68000 or 6809, FLEX, SK*DOS, CCF, OS-9, UniFLEX*
any object or source each - $50.00
any 3 object or source each - $100.00
Set of ALL object $200.00 - w/source $500.00

XASM Cross Assemblers for FLEX, SK*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.
*Complete set, FLEX, SK*DOS only - $150.00*

CRASMB from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8085, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK*DOS (binary). Written in Assembler ... e.g. Very Fast.

CPU TYPE - Price each:

For:	MOTOROLA	INTEL	OTHER	COMPLETE SET
FLEX9	$150	$150	$150	$399
SK*DOS	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 1632 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.
*FLEX, SK*DOS, CCF, OS-9/6809 $249.00*

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX, SK*DOS and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*
F, S and CCF - $79.95

supply

n. Optional regulation & heat sinks

o. Oscillator Modifications

p. 6809 Inputs (RESET, NMI, IRQ, FIRQ, HALT, DMA/BREQ, Etc.)

q. Memory modifications

r. I/O modifications and prototyping

s. Substituting the 6809E

As you can see from the above. Even the less expert among us can now try out his/her pet project without a lot of hassle. Like I said, *"Sure wish we had had this a few years back."*

The *PICOBUG* monitor contains functions allowing testing and debugging of hardware and software projects developed on this system. It has 24 I/O routines and 17 commands for communication between a terminal or host computer and the LAB6809 system. The commands for PICOBUG are:

1. ALTER - this is a register altering routine. Changes are made by placing the cursor under any nibble and typing in a hexadecimal value. The space bar skips to the next nibble, etc. This routine is a "ring routine" as it rotates around the register set. Rotation can be either forward or backwards. A C/R exits.

2. BREAKPOINT - a standard breakpoint routine which also provides a breakpoint table which may be displayed at any time.

3. CRAM - this function crams a block of memory with any desired byte.

4. DISPLAY - displays a memory map and does a non-destructive memory test. Testing is one byte per 2K block. This determines where RAM is located within the entire memory space.

5. EXAMINE - a standard hex-ASCII memory dump routine. Addresses and data are displayed 16 bytes per line.

6. FIND - memory is searched and every occurrence of a match is output to the terminal.

7. GO - used to continue after an interrupt or other PC halt. ALTER can place a new address on the stack and that will be the target for GO.

8. HELP - HELP is always on line and typing an "H" displays all the commands and their syntax.

9. I - J - K - these are reserved for pointers to user loaded subroutines or programs. Once vectors are loaded, simply typing one of these will immediately cause that routine or program to be executed.

10. LOAD - allows programs or data to be downloaded to RAM through the serial port. This is normally a tape input function.

11. MEMORY - a standard memory examine and change routine.

12. NEXT - this is a "nitty" function. It allows the last EXAMINE RAM location to be displayed after a single keystroke. After a RESET it displays the address range of the RAM tables.

13. OFFSET - calculates the offset between two entered addresses. Great for determining addresses of jumps and branches during debugging.

14. PUNCH - the output tape routine in Motorola S1 format. Remember paper tape, anyone?

15. QUICK - a destructive RAM test for a designated section of RAM. It is a "ring" type routine and it will continue forever, until aborted.

16. REGISTERS - retrieves the register values from the stack and display them to the terminal.

17. STACK - this displays a 64 byte block of memory as if the EXAMINE routine had been called with the current stack pointer as the beginning address.

18. TRANSFER - a block memory move routine.

PICOBUG stores all vector and variable tables in RAM for programmer access. Complete tables are provided to aid in the use of this feature. Also provided are 24 I/O routines that are fully documented and available for user application. Complete PICOBUG commented source is also provided.

All support software is written in 6809 FLEX and is provided on a standard FLEX disk, both object and source.

All the necessary "core" devices are provided on the board and each is socketed. The remainder of the board space is left for wirewrapping your project. Voltage lines are plainly marked on the wirewrap section of the board, which is handier than you might think.

This is an excellent thoughtout development system board. It can be used either on an S50 bus system or as a stand-alone system. I wish it had been available a few years back. If you have ever tried to fabricate a project on a standard perf-board, you can appreciate what a system like this can mean to the experimenter type. Hobbyist or professional, it is well worth the price, and more.

For ordering or additional information contact:

BARTON LABORATORIES
1606 W. 14th Street
Austin, TX 78703

(512) 477-6809

Advertised price: $395.00

A staff review

EOF

Pascal A Tutorial

By: Robert D. Reimiller
Certified Software Corp
616 Camino Caballo
Nipomo, CA 93444
805 929-1359

Editor's Note: This is the first of a series of tutorial articles on Pascal. The author is Robert Reimiller, an officer of "Certified Software Corp". They are the originators of OMEGASOFT™ PASCAL, a HLL that has long been available to the 68XX(X) community. Mr. Reimiller is the author of OMEGASOFT PASCAL. The latest versions of OMEGASOFT PASCAL are available for numerous 68XXX systems, including our Mustang-08A™ & Mustang-020™.

While written directly for the OMEGASOFT versions of Pascal, this series should be of value to all those programming in any version of Pascal. However, it should be understood that not all versions of Pascal have all the features that are standard with the OMEGASOFT versions.

OMEGASOFT PASCAL is a full feature implementation, and is available from our S.E. Media Division, Certified Software Corp or numerous dealers world-wide. Call or write for specific information. See S.E. Media advertising, this issue.

This begins a short series of articles designed to show the use of an industrial Pascal system, specifically, OmegaSoft Pascal. We will look at a number of short examples, starting with a simple "filter" program, and ending in a stand-alone control program. If there is enough interest there is the possibility of a more basic series on Pascal following this series. This series assumes the use of the native 68000 compiler, but much of it applies to the cross compiler which generates 6809 code using a 68000 host system.

For the purposes of discussion we will assume you have OmegaSoft Pascal installed on a OS-9/68000 system. This will provide you with the following main programs :

1) PC - the pascal compiler
2) RA - the relocatable assembler
3) LL - the linking loader
4) SE - the screen editor
5) PS - the pascal shell
6) LC - the linkage creator

You will notice that there is an editor, assembler, and linker included with the package, whats wrong with the Microware equivalents (R68 and L68)? First, this is an integrated package, all of the programs work together, this would not be practical using other manufacturer's utilities. Second, the Microware assembler/linker are designed with the 'C' language in mind, not Pascal. There are various subtle differences between the languages that require slightly different assembler and linker facilities to avoid compromises in operation.

It is possible to use "Scred" or any other editor available for OS-9, but you lose some of the integration of the package, which is designed to save your time, and after all, time is money.

As a practical example of a fairly simple program, let us write a program that will take Microware format assembly equate files and convert them to a form suitable for use as include files for the OmegaSoft assembler. The files we wish to consider are :

funcs.a process.a module.a io.a traps.a

To assign absolute values to a label R68 uses a "define offset" directive, and to indicate the current "offset" value, uses the symbol "." . RA uses the standard "define storage" directive and the symbol "*" to indicate the same thing when in absolute section (rather than code section). The other main item to be changed is converting "org <address>" in R68 format to "abs <address>" in RA format.

Our first example program :

```
program ex1 (Sv1.0 MW to OM include file
convertor)
                                        (input,
output) ;
  var
    labels, opcode : string[10] ;
    operand, comment : string ;
    inline : string[252] ;
    k : integer register ;
    dot_change : boolean register ;

  procedure getfield (var field : string ;
maxlength : integer) ;
    var
      i : integer register ;
      havefield : boolean register ;
    begin
      havefield := false ;
      field := '' ;
      for i := 1 to length(inline) do
        if inline[i] <> ' '
          then
            begin
              havefield := true ;
              exit
            end ;
      if havefield
        then
          while (i <= length(inline)) and
(inline[1] <> ' ') do
            begin
              if length(field) < maxlength
                then
                  append (field, inline[i]) ;
              i := i + 1
            end ;
      inline := substr(inline, i, 252 - i)
    end ;

  begin (ex1)
    writeln (' abs 0') ;
    while not eof do
      begin
        readln (inline) ;
        if length(inline) > 0
          then
            if inline[1] = '*'
              then
                writeln (inline)
              else
                begin
                  if inline[1] <> ' '
                    then
                      begin
                        getfield (labels, 10) ;
                        if labels[labels[0]] =
':'
                          then
                            labels[0] :=
labels[0] - 1
                      end
                  else
                    labels := '' ;
                  getfield (opcode, 10) ;
                  getfield (operand, 80) ;
                  comment := substr(inline, 1,
80) ;
                  dot_change := false ;
                  case upshift(opcode) of
                    'OPT', 'NAM', 'PSECT',
'ENDS' :
                      labels := concat('*',
labels) ;
                    'ORG' : opcode := 'abs' ;
                    'PAG' : opcode := 'page' ;
                    'DO.B' :
                      begin
                        dot_change := true ;
                        opcode := 'ds.b'
                      end ;
                    'DO', 'DO.W' :
                      begin
                        dot_change := true ;
                        opcode := 'ds'
                      end ;
                    'DO.L' :
                      begin
                        dot_change := true ;
                        opcode := 'ds.l'
                      end ;
                    'EQU' : dot_change := true
                  end ;
                  if dot_change
                    then
                      begin
                        k := index (operand,
'.') ;
                        if (k <> 0) and (operand
<> '''.''')
                          then
                            operand[k] := '*'
                      end ;
                  if (operand <> '') and
(operand[1] = '^')
                    then
                      operand :=
concat('$ffffffff-',
substr(operand,2,79)) ;
                  writeln (labels,' ',opcode,'
',operand,
                           ' ',comment)
                end
            else
              writeln
      end ;
      writeln (' code')
    end.
```

First of all, lets go through this program so you can get a feel for OmegaSoft Pascal. The program is setup as a "filter", as it reads all of of it's data from the standard input path "input", and writes all of it's output to the standard output path "output". These are specified in the program line so that the appropriate device drivers will be included. In the "var" section, we have a number of strings, this is not ISO standard Pascal, but most any real world pascal has this facility. The value in brackets after "string" tells the compiler the maximum length (in characters) of the string.

Another item worth noting in the variable declaration is the "register" directive following the last two declarations, these tell the compiler to store the variable in a CPU register rather than on

the stack. This produces faster and smaller code when accessing these variables.

Procedure "getfield" is responsible for parsing the input line and returning the next field in the line. A field is defined as a sequence of non-spaces, with any leading spaces skipped. There are 3 string related functions/procedures in the getfield procedure. The first is the "length" function, this returns an integer (16 bit signed) that corresponds to the number of characters that are valid in the string. This function is essentially the same as :

ord(inline[0])

This is because strings can be indexed from 0 to their current dynamic length. Element 0 is the dynamic length and indicates how many of the characters that follow are valid. It can range from 0 to 252. Indexing a string returns a character value. This feature is used in the "for" and "with" statements to scan through the string looking for blanks.

In the "for" loop we find the "exit" statement. The exit statement is used to leave a for, repeat, or while loop. If used without a parameter or with a parameter of one, it will exit the current loop. A statement such as "exit (3)" will exit from a loop 3 levels deep.

The second string related extension is the append procedure. It allows up to 16 string or character variables or constants to be appended onto a string. There is a related function in OmegaSoft Pascal called "concat", it will concatenate an unlimited number of string or character expressions to produce a string result, it is more flexible, but slower than append.

The third string function is "substr", this allows a part of a string to be returned, with specifications for the starting position and the maximum length of the returned string.

In summary, getfield will remove leading spaces and return a field in the parameter "field", with a maximum returned length of "maxlength". The input line has these spaces and the field removed so that the next time "getfield" is called, the next field will be returned.

The main program starts out by writing out " abs 0" which will make sure that the assembler is in absolute section, since the default at the start of assembly is code section. It then reads one line at a time until end of file. It first checks to see if it is a blank line, if so, it just writes a blank line. Next, it checks to see if the line is a comment (starts with an asterisk "*"), if so, it just writes out the line as-is.

If the first character of the line is a space, then there is no label and it sets the variable "labels" to a null string. If there is a label, it uses getfield to extract it into "labels". If the label ends with a ":", which indicates to R68 that it is a global label, it removes the ":".

It then uses getfield to extract the opcode and operand fields, with anything left assumed to be the comment field. The variable "dot_change" is set false, which will be explained later. A very useful extension in OmegaSoft Pascal is the ability to use strings in a case selector. We use the "upshift" function to allow both upper and lower case opcodes to be equivalent when we do the comparison.

The R68 "OPT", "NAM", "PSECT", and "ENDS" directives will not be needed, so we convert their lines to comments by placing an asterisk before the label field (if any) using the concat function. The "ORG" directive is changed to "ABS". The "PAG" directive is changed to "PAGE". The "DO.x" directives are changed to the equivalent standard "DS.x" directives.

For the "DO.x" directives and the "EQU" directives we set the dot_change variable true. This indicates that the operands for these directives are likely to contain the "." used by R68 to indicate the current offset. For this case we check to see if there is a "." in the operand field, and if so, we change it to a "*" to indicate the current location counter. A check is made to look for the period as a string literal "'.'", in which case it is not changed.

The last check is done due to a difference in usage of the "^" operator. R68 uses it for logical not, RA uses it for exclusive-or. Logical not is equivalent to subtracting the value from $ffffffff, so we substitute "$ffffffff-" for the "^". The new version of the input line is then written to the standard output by combining the label field, opcode field, operand field, and comments all separated by spaces.

To finish the output file, we return the assembler to code section by writing " code".

Next month we will enter the example program, debug it, and then produce a memory module that can be used as a utility.

OmegaSoft is a registered trademark of Certified Software Corporation, OS-9 and OS-9/68000 are trademarks of Microware Systems Corporation.

EOF

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

PRICOL.BAS

It is common experience that the difficulty in understanding (debugging, maintaining, modifying) a program grows faster than its size. How the code is printed in part make the task easier or on the contrary more difficult. For instance, I found that using the FLEX utilities P LIST was in this sense not so comfortable if my Assembler source was more than 4 or 5 pages.

The enclosed utility, PRICOL, was written in particular to deal with this sort of situation. It prints a text file typically a list of rather short strings such as an Assembler source or a BASIC program or the like by columns that read in natural order, that is the columns are left to right in a page while the lines are top to bottom in a column. Also, I use PRICOL for my documentation as it results in very compact printed pages and this contributes to keep under control the ever growing documentation deluge.

The program is written in XBASIC for an EPSON MX-80 F/T printer; it should be easily adapted to any other machine. Printing by columns as above explained may be done in principle very easily as follows: define a 2-dimensional string array, read the text file into it by columns and then print it from the array by lines, i.e. print the transpose matrix. The trouble with this scheme concerns the last printed page. This is because I set the rule for aesthetic reason that, if the text happened not to fill completely the last page, nevertheless the columns had to be as equal in length as possible. A recursive algorithm (l.270-l.290 in the accompanying program) takes care of defining the correct number of lines in the columns (in the array CL%()), given that IL% lines must be accommodated into NC% columns. How it works is illustrated by the following example.

Assume that it was required to print in 3 columns and that the line count for the last page falls between 30 and 33. 31 lines would be printed as:

```
EXA

      line #1    line #12   line #22
      .          .          .
      .          .          .
      line #10   line #21   line #31
      line #11
```

whereas 32 lines would give (notice how the numbering is altered in the 2nd and 3rd columns):

```
EXB

      line #1    line #12   line #23
      .          .          .
      .          .          .
      line #10   line #21   line #32
      line #11   line #22
```

Example

The "transposition" takes place in the next section, where of course CL%() provides the offset to retrieve the correct strings entering the current line to be printed L$ (1.350); notice that the condition in 1.330 insures that the last line is "padded" with null strings as needed if the column count is unequal.

The program otherwise requires little comments. Each page is read from a disk file into a string array LN$(), wide enough to accommodate a page of 72 (-6) lines by 4 columns. The page length is selected between 66 and 72 lines and the corresponding command is passed to the printer (1.90). Next, the file's name is input, along with the required number of columns in NC%; of course, the latter determines the length LG% of a string in a column, the string being truncated if its length is larger than LG% (see 1.260). The section 1.140-1.190 defines and determines the command to the printer for the appropriate tabbing, according to the selected number of columns. Additionally, CHR$(15) is also passed to the printer (1.150), which results in printing in compressed mode (the reason why WD was changed to 130 char./line in 1.40). Two error traps are provided (1.400-1.440); the first concerning error #4 needs no comment. The second is dealing with the EOF condition. The trap is normally visited twice: the 1st time, the flag F1% is armed, the page is processed with a line count IL% possibly less than normal and a form feed (CHR$(12)) is issued; only then the 2nd EOF terminates the printing.

```
10 REM "PRICOL"
20 REM VERSION 5, OCT. 1986
23 REM
25 REM BY J. LAVOREL
27 REM
30 DIMLN$(263),CL%(3)
40 EXEC,"TTYSET WD=130"
50 PRINTCHR$(12)+CHR$(15)
60 INPUT"HOW MANY LINES/PAGE (66-72)";FL%
70 IFFL%<>66ANDFL%<>72THEN60
80 OPEN"O.PRINT"AS0
90
PRINT#0,CHR$(27)+"C"+CHR$(FL%);:FL%=FL%-6
100 PRINT"DISK IN W!"
110 PRINT"FILE'S NAME (.EXT)
";:INPUTLINEF$
120 INPUT"HOW MANY COLUMNS(2-4)";NC%
130 IFNC%<2ORNC%>4THEN120
140 LG%=130/NC%:HT$=CHR$(9)
150 CP$=CHR$(15)+CHR$(27)+CHR$(68)
160 FORI%=1TONC%-1
170 CP$=CP$+CHR$(I%*LG%):NEXTI%
180 CP$=CP$+CHR$(0):LG%=LG%-1
190 PRINT#0,CP$;
200 ONERRORGOTO400
210 OPENOLDF$AS1:FI%=0
220 FORI%=0TOFL%*NC%-1
230 LN$(I%)="":NEXTI%
240 IL%=0:FORI%=1TOFL%*NC%
250 INPUTLINE#1,L$:IL%=IL%+1
260 LN$(I%-1)=LEFT$(L$,LG%):NEXTI%
270 FORI%=0TONC%-1
280 CL%(I%)=(IL%-1)/(NC%-I%)+1
290 IL%=IL%-CL%(I%):NEXTI%
300 FORJ%=1TOCL%(0)
310 L$="":I%=J%-1
320 FORK%=0TONC%-2
330 IFJ%>CL%(K%+1)THEN360
340 L$=L$+LN$(I%)+HT$
350 I%=I%+CL%(K%)
360 NEXTK%
370 L$=L$+LN$(I%)
380 PRINT#0,L$:NEXTJ%
390 PRINT#0,CHR$(12);:GOTO220
400 IFERR<>4THEN420
410 CLOSE1:PRINT"ERROR IN NAME OR EXT..."
415 PRINT"OR FILE NOT FOUND...":RESUME110
420 IFFI%=1THENRESUME450
430 IFERR<>8THENONERRORGOTO
440 FI%=1:RESUME270
450 CLOSE1
460 PRINT"ANOTHER FILE (Y-N)
";:INPUTLINER$
470 IFR$<>"Y"ANDR$<>"Y"THENCLOSE0:END
480 PRINTCHR$(12)+CHR$(15):GOTO100
```

Jean LAVOREL
"Santoline" Qu. Bigourdin D13
13770 VENELLES France

EOF

Continued From Last Month

XBASIC Xplained

or

Things you won't find in the documentation

NUMBER BASE CONVERSIONS

Just for a break, we'll take another look at our Decimal-to-HEX conversion program of earlier on. Did anyone observe that the self-same routine can be used to convert from decimal to ANY base in the range 2 - 63? Though there'll be some pretty weird-looking digits beyond base 27! All that has to be done is to replace the two occurrences of '16' in Line 30 with the desired base-number. Of course, if the maximum base-number is restricted to 10, the section which adjusts by 7 for digits in excess of 9 will not be needed. To really round off the program, Line 10 should be changed to read :

```
10  H$ = "": INPUT "Enter decimal-number and base ",D,B%
```

and change '16' to 'B%' in Line 30.

While I'm on the subject, it might not be a bad idea to give the reverse program, that is, to convert from any base to decimal, so here goes with a quickie :

```
10  D = 0: INPUT "Enter number and its base ",N$,B%
20 REM CONVERT NUMBER-STRING IN ANY BASE TO A DECIMAL
30  J% = LEN(N$): FOR I% = 1 TO J%
40    K% = ASC(MID$(N$,I%,1) - 48:  IF K% > 9 THEN K% = K% - 7
50    D = D + K%: IF I% <> J% THEN D = D * B%
60  NEXT I%
70  PRINT D: GOTO 10
```

Using logic functions instead of IF-THEN, Line 40 could be re-written so :

```
40    K% = ASC(MID$(N$,I%,1) - 483 3+ 37 3* 3(3K3% 3> 393))
```

Note that I use 3+37 this time because I want to subtract 37 if (K% > 9) is TRUE, ie it becomes -1. See if you can similarly combine Lines 40 and 50 into one line! (See next paragraph for solution) Before leaving this subject, and returning to the main stream, it might be worthwhile mentioning that to convert from Base 1 to Base 2 (and get the decimal equivalent along the way) the two programs should be chained. That is, convert from Base 1 to decimal, and then straight on to convert decimal to Base 2. You'll find that routines which use power-functions like a^b to achieve base-conversions are not only much slower to execute, but are apt to become a little unreliable, as they sometimes use logs and antilogs in their calculations.

SOLUTIONS TO ABOVE PROBLEMS

In the previous paragraph the reader was given a not-so-easy problem to think about. Maybe you should have a go at it before reading on! We were asked to convert the following :

```
50  D = D + K%: IF I% <> J% THEN D = D * B%
```

into a single-statement line by using logic-functions instead of IF-THEN. Essentially, what we have to do is to multiply the new value of 'D' (as it exists at the point 'IF') by 1 if I% = J% and by B% if I% <> J%. This is accomplished by the following line :

```
50  D = (D + K%) * (1 - (B% - 1) * (I% <> J%))
```

Here we see our original first statement enclosed in parens, and note that it is to be multiplied by '1'. This '1' is to be further modified by the (B% - 1) factor 'IF I% <> J%)', which is how we read the final portion '* (I% <> J%)'. Earlier discussions will have taught how this will evaluate to '0' if 'I% <> J%)' is FALSE (ie I% IS EQUAL TO J%), and to '-1' if it's TRUE. So, (B% - 1) will be multiplied by '0' in the first instance, and by '-1' in the second, giving an ultimate result of either (B% - 1) * 0, that is '0', or (B% - 1) * -1, that is '-B% + 1'. To complete the math then, our (D + K%) will be multiplied either by (1 - 0) or by 1 - (-B% + 1), which evaluates to 1 in the first case and to '1 + B% - 1', or 'B%' in the second. And hey presto! we've arrived. Hope this hasn't frightened you off, but it does enable us now to tack Line 50 on to the end of Line 40, and even Line 60 onto the end of that lot, if we so desire. All of which will also save a little memory for us.

GETTING INPUT FROM THE KEYBOARD

There are actually four different ways we can request input from the keyboard. Do you just use 'INPUT' as a matter of course, or do you sometimes consider the advantages (or disadvantages) of 'INPUT #0', or 'INPUT LINE' or even 'INCH$(0)'? Let's look at each in a little more detail ....

INPUT Fairly straightforward. It can be taken neat, as in

'INPUT I%' or 'INPUT I%,J$'

for example, or it can print an input request message of some kind, such as

```
50  INPUT "Please enter your name and age",N$,A%
```

In all cases it will display a '? ' (note the SPACE after the '?'), and pause for a response terminated by a C/R, with commas to separate multiple responses. In the case of multiple requests,if the required number of responses is not made, further '? 's will be displayed until INPUT is satisfied. On the other hand, extra responses will be ignored! INPUT will not accept a C/R as a valid response. The function of C/R is to indicate 'End of reponse'.

Note, too, that a '?' on the end of "Please enter your name" is not appropriate, as we are not asking a question here, but issuing an instruction, whereas it would be OK on the end of "What is your name". So we come to

INPUT #0 which is almost the same as INPUT, except that it does not print a trailing '? '. Its form is

```
50  INPUT #0, "Please enter your name ... ",N$
```

There are some differences, however, in that when you enter your name, followed by C/R, the cursor does not move down to the following line, but simply returns to the left-hand margin. Should this input request then be followed by another, such as 'INPUT #0, "Your age ... ",A%', (after a response of, say, BOB to the first) the first message would be overlaid and you'd see (on the self-same line) the request :

Your age ... your name ... BOB

with the cursor positioned over the 'y' of the second 'your'.

To correct this (unless you're using direct cursor-control to position your messages on the screen) you should follow your 'INPUT #0' request with a 'PRINT', thus

```
50  INPUT #0, "Please enter your name ... ",N$: PRINT
```

which will cause the next message to be displayed on the succeeding line.

INPUTLINE

The XBASIC article tells us that this form is used to INPUT an entire line, including embedded SPACEs, etc. No messages can be displayed (as with INPUT and INPUT #0), and only one variable-name may be listed :

```
50  INPUT LINE A$     or     50  INPUT LINE B1$(5)
```

It displays a '? ', just as does INPUT, and normal testing of this form would indicate that there is no apparent difference between the two (apart from the restrictions mentioned), as INPUT will also accept a line of text as a response. There is one other difference though, which is not mentioned in the article, and that is - INPUT LINE will accept a mere C/R as a valid response, whereas the other two will not. They'll just keep coming back with further '? 's until satisfied. This difference could be useful in cases where you wish to set up a default response (default to C/R) as in :

```
50  PRINT "Is your Printer a DOT-MATRIX* or THERMAL";: INPUT LINE R$
60  IF R$ = "" GOTO 1000 ELSE 2000
```

where the '*' indicates the default response. Line 60 checks for a C/R by comparing R$ with the NUL-string, ie no data entry. Observe that because INPUT LINE does not allow a message to be embedded, we cause it to be displayed by means of the PRINT statement, and then follow on with the INPUT LINE statement. Note the ';' immediately following the request message. This to nullify the C/R which would normally occur on completion of the PRINT statement, so now the cursor is held at the end of the message, waiting for a response.

INCH$(0)

This is another form which does not allow embedded messages. Unlike INPUT LINE, a single-character response without a following C/R is sufficient to meet its needs. Of course, the single response may itself be a mere C/R! A very useful means of getting responses of the 'Y/N' type, or single-letter commands in a game, for instance. It is entered in the following pattern :

```
50  PRINT "Do you like this? (Y or N) ";: R$ = INCH$(0)
60  IF R$ = "Y" GOTO xxxx ELSE GOTO yyyy
```

INCH$(0) does not put out a '? ', and just like INPUT #0 it does not move to a different line after accepting input. So here again, unless you are using direct cursor-positioning, you would follow the INCH$(0) with a ': PRINT'. In leaving this subject for now, you should again note the ';' following the request-message. Its function is exactly the same as that described for INPUT LINE. Another difference between INCH$(0) and the other forms of INPUT is that if the BREAK key is hit while waiting for INCH$ input, the program will BREAK without any hesitation, but will refuse to recognise the CONT instruction afterwards.

Finally, note that the '0' in parens serves to indicate to XBASIC that input is to come from the User's terminal. If, on the other hand, Channel 1 has been OPENed for input, then INCH$(1) would input a single character from the Data-File on Channel 1.

LSET & RSET

I've never seen these used in normal XBASIC programs, other than in connection with FIELD statements where input is from a Data-File on disk. It seems that the average programmer is content with a restricted set of Statements, and just doesn't feel the need to explore new territory, even though there may be enormous benefits awaiting him/her out there! Let's talk a little about these two, shall we?

Let me begin by saying that the reason they are not normally encountered in 'ordinary' XBASIC programs is probably because they would serve no purpose in programs which scroll continuously. This will become clear in the discussion following.

LSET Let's suppose we have a screen-oriented display with an area reserved for error-messages of various kinds, and that a message "Illegal move entry" has just been displayed. Now a new message has to be overlaid in the same area, say "Syntax error". The problem is that if it's simply overlaid we'll see the message "Syntax error entry", so obviously we have to delete the old message in some way. This, of course, leads us into the different methods which could be employed. For example :

a. If the message line extends right to the edge of the screen one could simply position the cursor at the start of the message, issue an "Erase-to-end-of-Line" code, and write in the new message.

b. If the message area has some vital part of, let's assume, a game-board to the right of it, we can't use (a) above as it would wipe out part of the game-board too. In this situation we could position the cursor as above, print maybe 30 SPACEs, re-position the cursor, and then write the new message, OR

c. We could use LSET. To do this nicely, we would, by way of a demo program, write :

```
10 A$ = "                    "    (just 20 SPACEs)
20 INPUT Q$: LSET A$ = Q$
30 PRINT A$; "X": GOTO 20
```

RUN it, and try entering responses of up to 20 characters, or more, in response to the prompt. Note that no matter how long or short the response is, LSET will *always* pad out to 20 characters with SPACEs, before displaying the "X" (which is included simply to mark the end of A$). If the response is over 20 characters long, LSET will truncate it down to 20 characters. There is another little advantage to using LSET, and that is that having printed our message, the cursor will be nicely positioned at our point "X", ready for whatever has to follow next - possibly to request a new and acceptable input this time around - thus saving a possible TAB(XX) statement.

RSET What can I say about this? Except that whereas LSET *left*-justifies the message, RSET *right*-justifies it, padding from the left with SPACEs. This would be more useful for displaying 'cash' entries, as the amounts would be neatly columnised at the right.

A DIGRESSION

I'm now going to digress, and discuss XBASIC a little more generally. First, another error in the documentation, which states that the maximum length of an XBASIC line is 127 characters. Not true! Lines can be up to 255 characters in length.

Secondly, an undocumented feature, namely that if one enters LIST -100 in response to the READY prompt, your program will be LISTed from the very beginning up to and including Line 100. The reverse, LIST 100-, does *not*, however, LIST from Line 100 to the end of the program, though this feature *does* work in RBASIC (Micronics Research Corp's expanded in scope, albeit much shorter in length, and much more powerful successor to XBASIC).

Thirdly, and much more importantly, Dr. L. P. L. Piacenza, of the University of Transkei, Southern Africa, has pointed out to me a **very** serious flaw in XBASIC's floating-point math operations, which can produce **completely** erroneous results when dealing with very large, or very small, numbers. The intent of XBASIC is that **any** number, if continually increased, should eventually run into an OVERFLOW condition, and be trapped as such with an appropriate error-number. Similarly if the number is continually decreased it should eventually run into an UNDERFLOW condition. The following test program indicates quite clearly that this is not always so, as XBASIC will very often reset to '0' the number which has OVER/UNDERFLOWed, **without trapping it as an error-condition**, and allow the program to continue with its now erroneous calculations. This is not so serious with UNDERFLOW, as an infinitesimally small number can be regarded as '0' for most purposes, but in the case of exceptionally large numbers, gross errors will be produced in the final result, which sometimes may look sufficiently 'correct' as to cause its acceptance by the User. Here's the test program to demonstrate the effect :

```
10 ON ERROR GOTO 70
20 Y% = 1
30 X = Y%
40 X = X * 2: IF X = 0 then PRINT CHR$(7);: GOTO 60
50 X = X * 5: IF X = 0 THEN PRINT CHR$(7); ELSE GOTO 40
60 PRINT Y%;: Y% = Y% + 1: GOTO 30
70 IF ERR = 101 THEN PRINT "XXX ";: Y% = Y% + 1: RESUME 30
```

This program commences with the number 1, and keeps looping, multiplying it first by 2, then by 5, until eventually XBASIC either correctly traps it as an OVERFLOW or incorrectly resets it to '0'. In the first event the 'ON ERROR' instruction sends program-flow to Line 70 (causing a display of 'XXX '), but if '0' is detected the program will BEEP and display the offending number. In both cases the initial 'seed' number is then bumped by one and the process repeated, filling the screen with a pattern such as :

```
1 2 XXX XXX 5 6 7 XXX .....
```

thus pinpointing those numbers which are not correctly trapped as OVERFLOW. However, if a different pattern of multiplication were installed, let's say multiplying by the sequence X*2 and X*3, a different pattern of results would be obtained. To test for incorrect *UNDER*RFLOW detection, simply replace '*' with '/' in lines 40 and 50. You'll be simply amazed at how many numbers escape detection!

The fix is to locate in XBASIC the following code-sequence (somewhere around address $37C3, depending on your particular version), and to replace ALL that code with NOPs, that is, '12's. Address $37C3 is valid for Version 24 of 6809 XBASIC. The fix for UNDERFLOW involves the insertion of an extra 2 bytes at a different location, and is therefore not easy to implement, unless you have already disassembled XBASIC and know your way around the code. In any case, as mentioned earlier, UNDERFLOW is not of such consequence as OVERFLOW when reset to '0'. Here is the code-sequence to look for to correct the OVERFLOW problem :

```
Address $37C3   A6 C833
        E6 47
        1083 0000          13 bytes to be changed to '12'
        27 07
        2B 05
```

The patch for the UNDERFLOW problem is to locate at address $3804 (XBASIC version 24) the code-sequence 'C3 0081' and to insert immediately after it (not overwrite) the code '27 C7' to cause a branch back to address $37D0, and thus to a correct UNDERFLOW indication. The code at address $37D0 is '86 65', ie, 'Load accumulator A with error-number 101'. Afterwards it will be necessary to re-assemble your XBASIC.

GESPAC, INC

50 W. Hoover Ave.
Mesa, AZ 85202
(602) 962-5559
Telex 388 575

Dear Mr. Williams,

A few months ago, we announced that GESPAC will distribute the G-64 product line of MPL-AG in the United States.

Today I am pleased to introduce two new and exciting products from that company. The first board is a 16 channel relay mode; the second a quad channel, multi-mode serial interface. Both cards are ideally suited for application industrial environments.

I would be grateful if you could help us share this information with the readers of your publication. Enclosed is a press kit we prepared to that end.

Please feel free to contact me if you have any questions or need more information.

Sincerely,

Cosma Pabouctsidis
President

CP/t
Enclosure

Editorial Contact:
Cosma Pabouctsidis
GESPAC Inc
50 W. Hoover Ave
Mesa, AZ 85202
(602) 962-9559

Sales Contact:
Mark Stephens
GESPAC Inc
50 W. Hoover Ave
Mesa, AZ 85202
(602) 962-5559

FOR IMMEDIATE RELEASE

MPL INTRODUCES INDUSTRIAL I/O BOARD SET

FOR THE G-64 BUS

MESA, AZ, May 1, 1987–MPL, a board level product manufacturer for the G-64 bus, introduces two boards geared to industrial process control applications. Both cards are built on a single height Eurocard of 100 by 160 millimeters and are compatible with the standard G-64 bus.

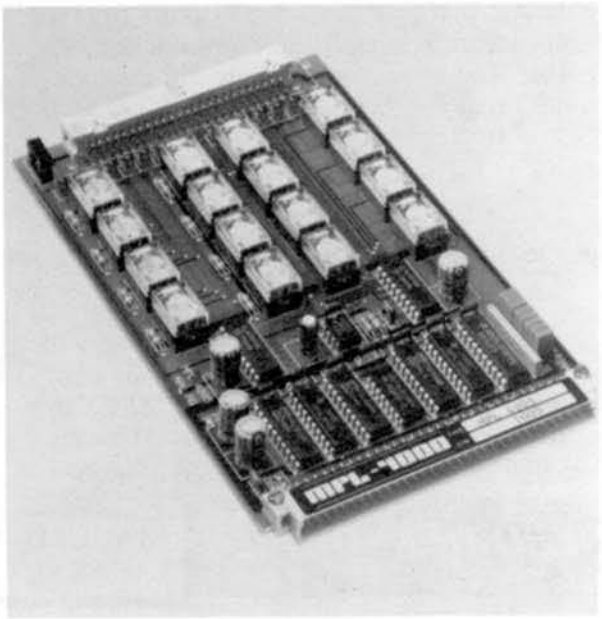
The MPL 4205 is an interface card with 16 relays, the MPL 4210 is a universal, quad channel RS 232, RS 422 and current loop serial communication board. The G-64 bus is an easy-to-interface, 16-bit bus aimed at real-range industrial process control applications.

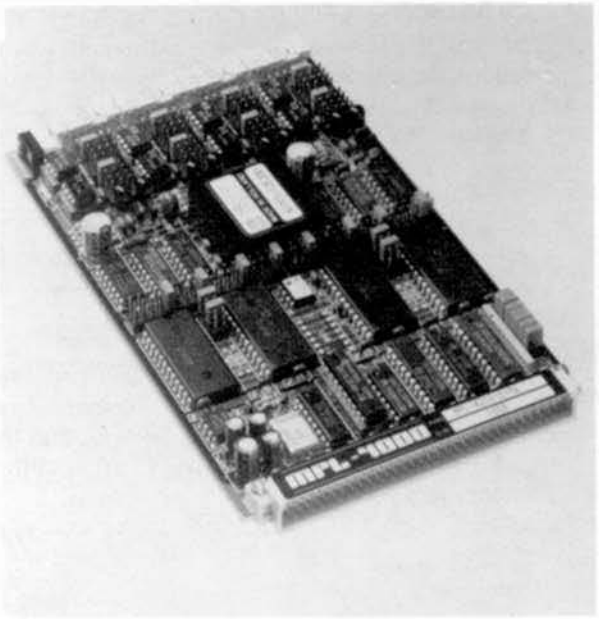
Each relay of the MPL 4205 has two contacts, direct and inverted, capable of driving 1 Amp at 24 VDC, or 500 mA at 100 VAC. The board is equipped with 16 LEDs, each indicating the state of its assigned relay.

The MPL 4210 handles four asynchronous channels at data rates of 300 to 19200 baud. The output interface of the board can be ordered in one of three basic configurations: RS-232 only, RS-232 and 422, or RS-232, 422 and 20 mA current loop. The RS-422 and current loop mode are highly immune to noise and allow communication up to 3,000 feet at 9600 baud. An on-board DC/DC converter provides total voltage insulation on the communication channel.

The MPL 4205 is available today at $425 in single quantities. The MPL 4210 is available from $395 to $875 depending on the configuration.

MPL has its headquarters in Baden, Switzerland and is represented in the U.S. by GESPAC Inc.





MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8698

For further information contact:

EDITORIAL CONTACT:
Mark Vercruysse
512/928-6804

INQUIRY RESPONSE:
Technical Information Center
P.O. Box 52073
Phoenix, AZ 85072

READER CONTACT:
Dean Mosley
512/440-2839

MOTOROLA INTRODUCES THE 25 MHZ EMULATOR MODULE FOR THE MC68020

Austin, Texas, May 28, 1987... Motorola's Microprocessor Products Group introduces a new model of the MC68020 Emulator Module for the powerful MDS-300 family of development system products. The MDS-300 development system is the only MC68020 Emulator available with speeds greater than 20 MHz.

The new Motorola Emulator Module makes leading edge 25 MHz microprocessor based product development available to product designers both large and small; letting developers with nearly any resource and at any scale to enter state-of-the-art 25 MHz product development for powerful 32-bit designs.
The MC68020 Emulator Module duplicates in real-time all the functions provided by the MC68020 MPU so that the MPU chip can be replaced in the user's target system under development. The Emulator Module operates in conjunction with the MDS-300 Control Station, the interface between the Emulator Module and a host computer system (see attached diagram). The MDS-300, in standalone mode, is capable of downloading target application software from almost any host.

Key features of the new Emulator include:

- Zero wait state emulation for user target memory at 25MHz
- Zero wait state emulation for internal memory at 20MHz and a maximum of one wait state at 25MHz
- Three hardware and 64 software breakpoints
- Elaborate debug/display structure
- Unrestricted use of full 4 Gigabyte address space
- User control of selected MPU and target signals
- Multiple emulator synchronization
- Interface to optional system performance analyzers

The module is available in three different emulation memory sizes: 64K, 256K, or 1 M byte. This memory can be used for overlaying target memory at any address; variable wait state assignment to match target memory speed; variable bus size selectability to match target memory size; and write protection.

Three classes of self-tests are provided to verify operation: power-up, warm-start, and extended. Each test performs successively more comprehensive self verification of functionability.

Part Number	Description	Price Qty (1-5)
MC68020HM3C-4	64K Emulation Memory	$ 8,900
MC68020HM3C-5	256K Emulation Memory	$ 11,500
MC68020HM3C-6	1 Meg Emulation Memory	$ 14,400

For more information contact your local Motorola Sales Office.

WINDRUSH
Micro Systems Ltd.

Windrush Laboratories (Reg. Office)
North Walsham, Norfolk NR28 9SA
Tel (0692) 404086
Telex 975548 WMICRO G
MARCON approved consultants

68 Micro Journal
Attn: Dan Williams Snr., Editor
5900 Cassandra Smith Road
Hixson, TN 37343
U.S.A.

Your Ref -
Our Ref WCD/jd-337
Date 11/5/87

STAND-ALONE DATA ACQUISITION SYSTEM IS USER PROGRAMMABLE

Windrush Micro Systems Limited are pleased to announce the immediate availability of high capacity 19" rack mounting data acquisition and control systems which offer the capability of being user programmed.

The products feature a choice of micro-processor and over 1200 isolated digital channels in a single rack. All inputs and outputs are optically isolated to suit the demands of the harsh electrical environments found in industrial sites. The systems are also available in IP-66 enclosures.

The systems can be supplied as fully programmed data acquisition or process control systems or as hardware modules which the end user may program himself. Windrush will also customize the software package to suit individual requirements.

The systems also offer PL/9 as an in-built control language as an option. This PASCAL like programming language has been optimized to suit the needs of real-world real-time process control systems.

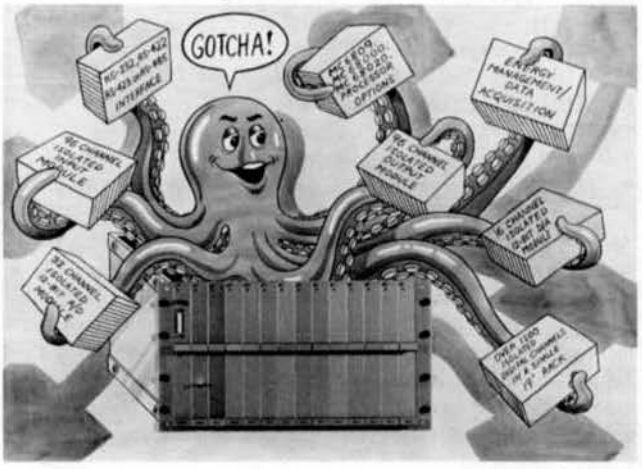
The systems may be networked together over an RS-422/RS-485 interface which may extend over several kilometers.

Applications include data acquisition, process control and energy management. Features include:

- Choice of 6809, 68000, 68010 or 68020/68881 processor modules.
- 4 channel RS-232/RS-422/RS-423/RS-485 interface module.
- 96 channel optically isolated digital input module.
- 96 channel optically isolated digital output module.
- 32 channel optically isolated 12-bit analogue-to-digital (A/D) module.
- 16 channel optically isolated 12-bit digital-to-analogue (D/A) module.
- Over 1200 isolated digital channels in a single 19" rack.
- Available in IP-66 enclosures for distributed processing applications.

Directors: J B Daly, W C Dickinson (U.S.)

Registered in England and Wales No. 1611025 Vat Reg. No. 372 9531 33



Microplex

RFM MICROPLEX SYSTEMS Ltd.
265 East 1st Avenue
Vancouver, B. C. V5T 1A7 Canada

Vancouver: (604) 875-1461
Toronto:(416) 283-2487
Telex: 04-352848 VCR

New Product Announcement

RFM Microplex Systems is pleased to announce the release of the M108 Dual Serial Interface board. The M108 is designed for use with computers equipped with an SS30 bus and the OS9 operating system. The product was developed to eliminate the frustration caused by disassembling computers to change baud rates and the desire for 19.2KBPS operation.

The M108 is designed around the Signetics SCN2681 dual serial interface IC which features progammable baud rates, quadruple buffered recieve data registers, integral flow control (programmable option) and variable length stop bits.

The SS30C interface is buffered with HCMOS devices to minimize loading and drive heavily loaded data buses. The RS232C interfaces use CMOS driver/reciever chips and each interface may be configured as a DTE or DCE type device.

The board is supplied with software drivers from Morse Softcode for the OS9 operating system. Two versions of the drivers are supplied, one that is equivalent to standard MC6850 type interfaces and one that takes advantage of the SCN2681's advanced capabilities. A utility ("Zmode", also from Morse Softcode) which is similar to the Tmode and Xmode programs is also included.

Microplex is a 9 year old manufacturer of data communications equipment. Our current product line includes (partial list):

- M108 dual serial SS30C bus interface
- a complete line of RS232C and centronic A/B switches
- M103 Code Activated A/B Switch
- M106, M107 and M112 breakout boxes
- M102 modem eliminator
- M104, M116 and M128 modem/port sharing devices
- M110 analog line sharing device
- M122, M123 async to sync and NRZ to NRZI converters

At Microplex, we specialize in building products that combine exceptional quality with competitive pricing. All of our products undergo extensive testing and stringent quality control including a 5 day environmental test period to ensure maximum performance and lasting reliability. To assure further confidence in our products, each unit that we manufacture is backed by a two year parts and labour warranty in the unlikely event of a failure occuring.

Our on-going research and development efforts continue to produce state of the art products that increase efficiency and cost savings for data communications users.

OS9 is a trademark of Microware Systems Corp.

CERTIFIED SOFTWARE CORPORATION

616 CAMINO CABALLO, NIPOMO, CA 93444 USA TELEPHONE 805-929-1395 TELEX 467013

INTEGRATED CROSS PASCAL PACKAGE FOR 6809 PROCESSORS

Nipomo, CA., May 12, 1987 - Certified Software announces a new Pascal compiler system designed to run on a 68000 host system and generate code for a 6809 target.

The system features a menu driven "Pascal Shell" that allows organization of project files and control of the other programs in the package. The shell allows access to the screen editor, compiler, assembler, linker, and target debugger. All source files are available in a menu for the editor. Only pascal source files are shown in the compile menu, likewise, only assembly source files are shown in the assemble menu.

A unique feature of the supplied screen editor is the ability to compile or assemble directly from the editor's internal buffer, greatly enhancing the speed of syntax checking. The compiler and assembler also report back errors to the editor, resulting in automatic placement of the cursor for error correction.

The pascal language supported is based on the ISO standard and has extensions for real time applications. A few of the major extensions include additional data types and operators, modular compilation, interrupt and task procedures, and selective error trapping.

This package is available to run on OS-9/68000 based host computers. It is available directly from Certified Software, or through one of it's European distributors. Priced from $700.

OmegaSoft is a registered trademark of Certified Software Corporation. OS-9/68000 is a trademark of Microware.

OMEGASOFT™ PASCAL

Classifieds

As submitted - No Guarantees

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900 Qume Sprint 5 - $800

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply. New in box. $125 each.

5 - Siemens 8" Disk Drive , $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.

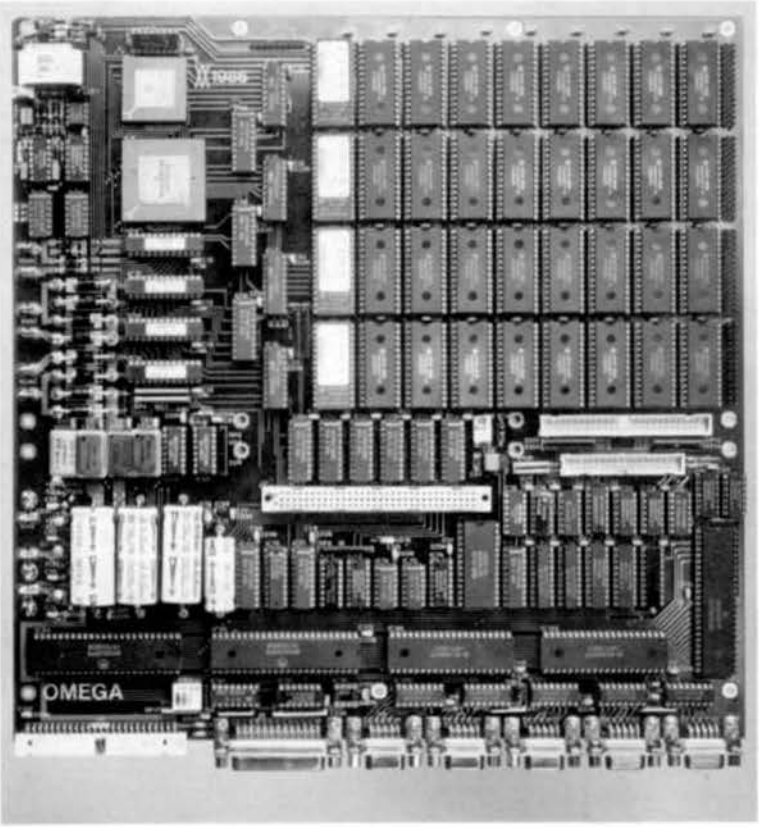
SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Paralell Port, MP-09CPU Card - $900 complete.

(615) 842-4600 M-F 9AM to 5PM EST

•••

GIMMIX - Dual D550, cabinet and power supply - $100.00 ; DMA Controller #68 - $95 ; 6809 Plus, OS9 - GIMX, etc. - $95; Make Offer : 2 part serial, band, parallel, filter, Qume floppies.
Ed 303-278-4455 or 303-233-8231 (evenings)

CT-82 terminal with Sardis Technology SBC built in. 64K with FDC, serial and parallel ports. Runs Flex and OS-9 $395.00, MF68 dual disk drive $125.00.
Phone - (515) 276-2182



12" x 12"

On-board power supply . . . only requires external 10 VA transformer.

5 x RS-232C serial ports.

SCSI initiator.

FDC controller

Parallel printer port.

16-bit bi-directional I/F

Non-volatile clock calendar

Fully buffered bus expansion

QUANTITY DISCOUNTS
2-5 less 5%, 6-9 less 10%
10-24 less 15%, 25-99 less 20%
100+ less 25%

12.5 MHz MC68020 32-bit processor and 12.5 MHz MC68881 Floating Point Coprocessor plus 1 Mb non-volatile, zero wait-state 100% STATIC RAM all AS STANDARD!

Ωmega . . . The OEM's CHOICE

OMEGA/12.5	OMEGA w/12.5 MHz 020/881 & 1 MB RAM	$2750.00
OMEGA/16.6	OMEGA w/16.67 MHz 020/881 & 1 MB RAM	$3295.00
OMEGA/LTX	OMEGA 120/240 VAC Line Transformer	$ 22.00
OMEGA/1MB	1 megabyte additional STATIC RAM	$ 895.00
OMEGA/SER9	9 port RS-232C serial expansion board	$ 645.00
OMEGA/GRF	640 x 480 x 4 bits/pixel ACRTC graphics I/F	$ 925.00
OMEGA/OS9	OS-9/68K™ 'Professional' configured for OMEGA	$ 595.00

PRICES INDICATED INCLUDE SHIPPING AND APPLY TO U.S. CUSTOMERS ONLY

VISA/MASTER CARD ORDERS ACCEPTED

OS-9 is a trademark of Microware Systems Corporation

NON-U.S. CUSTOMERS SHOULD CONTACT THE NEAREST DEALER FOR PRICE & DELIVERY INFORMATION

NORTH & SOUTH AMERICA

LLOYD I/O INCORPORATED
19535 NE GLISAN
P.O. Box 30945
PORTLAND, OR 97230 (USA)
TEL: (503) 666-1097
TLX: 9103805448 LLOYD I O

ALL OTHER ENQUIRIES

WINDRUSH MICRO SYSTEMS
WORSTEAD LABS.
N. WALSHAM, NORFOLK
NR28 9SA, ENGLAND
TEL: (0692) 404086
TLX: 975548 WMICRO-G

CONTINENTAL EUROPE

SNIJDER MICRO SYSTEMS
SCHOOTEINDSEWEG 8a
5756 BD Vlierden
The NETHERLANDS

TEL: (0)4930-11975-13666

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language ProGrams and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVEC1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINTC3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included in ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

S. E. Media

Savings to 50%
And More!

All other items **10%** off - Sale ends 7/31/87
Add: $2.50 S/H all software, each item.

ITEM	Prices Reg	Special
CoCo/OS-9 BASIC Xref w/source	79.95	49.95
CoCo/OS-9 BASIC Xref w/o source	39.95	24.95
OSM - 6809 Assembler Lloyd I/O	99.95	49.95
Search & Rescue - Lloyd I/O OS-9	35.00	24.95
C COMPILER - Windrush FLEX	295.00	195.00
BIZPACK Vairginia Co. FLEX		
VIRTUAL TERMINAL CoCo/OS-9	49.95	36.95
CoCo KANSAS CITY BASIC	39.95	24.95
LSORT OS-9 Sort/merge I II	85.00	54.95
BAS-EDIT FLEX (TSC BASIC util)	39.95	19.95
ASSEMBLERS Assorted Lloyd I/O	discounted	call
K-BASIC Compiler compiles XBASIC - FLEX - OS-9		
very popular BASIC compiler..	199.95	79.95
DYNACALC Great spreadsheet	200.00	99.95
DYNAMITE FLEX OS-9 CoCo all	varies	49.95
STYLO Word Pro CoCo OS-9	99.95	49.95
PL9 5" FLEX	198.00	129.95
SCREDITOR III 5"	175.00	119.95

AR Great Plains FLEX	295.00	129.50
RRMac Great Plains FLEX	150.00	74.95
INFOMAG Great Plains UniFLEX	395.00	99.50
SPELLING CHECKER FLEX	99.95	49.95
MAIL MERGE FLEX	79.95	39.95
C Compiler Introl FLEX - OS-9	575.00	295.00
REL MACRO ASMB Microware OS-9	200.00	99.50
CHECK & TAX Star-Kits FLEX	50.00	19.95
STAR-DOS CoCo	34.50	19.50
HUMBUG Star-Kits (Tape)	40.95	9.95
REMO-TERM CoCo (tape) Remote CRT ?		9.95
SPELL 'N FIX Star-Kits FLEX	89.29	39.95

ASSORTED TSC 6809 FLEX UniFLEX 40-75 % off

Assorted items: (some new - some used working)
SWTPC SO-9 Computers 128K 3000.00 75% off
Assorted boards, etc. up to 90% off - call

NOTE - STAR-DOS SK-DOS runs same as FLEX

See S.E. Media Catalog - page 29 for descriptions

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE
NOW SK*DOS available

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP

A Division of
Computer Publishing, Inc.

5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

6809<>68XXX UniFLEX

X-TALK

A C-MODEM/Hardware Hookup

Exclusive for the MUSTANG-020 running UniFLEX, is a new transfer program and cable set from DATA-COMP (CPI). X-TALK consist of 2 disks and a special cable, this hook-up enables a 6809 SWTPC UniFLEX computer to port UniFLEX files directly to a 68XXX UniFLEX system.

This is the only currently available method to transfer files, text or otherwise, from a 6809 UniFLEX system to a 68000 UniFLEX system, that we have seen. A must if you want to recompile or cross assemble your old (and valuable) source files to run on a 68000 UniFLEX system. GIMIX users can directly transfer files between a 6809 GIMIX system and our MUSTANG-020 68020 system, or GIMIX 68020 system. All SWTPC users must use some sort of method other than direct disk transfer. The 6809 SWTPC UniFLEX disk format is not readable by most other 68000 type systems.

The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 DB25 connectors. A special SWPTC+ cable and software is also available, at the same price. Orders must specify which type SWTPC 6809 UniFLEX system they intend to transfer from or to.

The X-TALK software is furnished on two disks. One 8" disk containing the 6809 software and one 5" disk containing the 68XXX software. These programs are also complete MODEM programs and can be used as such, including X-on X-off, and all the other features you would expect from a full modem program.

X-TALK can be purchased with/without the special cables, however, this SPECIAL price is available only to registered MUSTANG-020 owners.

X-TALK, w/cable $ 99.95
X-TALK only 69.95
X-TALK w/source $149.95

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

Telephone 615 842-4601
Telex 510 600-6630

Note: Registered MUSTANG-020 owners must furnish system serial number in order to buy at these special low prices.

68 MICRO JOURNAL
Reader Service Disks

Disk- 1 Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Diskexp, *Disksave.
Disk- 4 Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6 **Purchase Order, Index (Disk file indx).
Disk- 7 Linking Loader, Rload, Harkness.
Disk- 8 Crtest, Lanpher (May 82).
Disk- 9 Datecopy, Diskfix9 (Aug 82).
Disk-10 Home Accounting (July 82).
Disk-11 Dissembler (June 84).
Disk-12 Modem68 (May 84).
Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help, Date.Txt.
Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib
Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17 Match Utility, RATBAS, A Basic Preprocessor.
Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Annstrong), CMDC ODE, CMD.Txt (Sept. 85 Spray).
Disk-19 Clock, Date, Copy, Cat. PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21 Utilities & Games - Date, Life, Madness. Touch, Goblin, Starshot, & 15 more.
Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23 ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March1986.
Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25 KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26 Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27 ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28 CT-82 Emulator, bit mapped.
Disk-29 **Star Trek
Disk-30 Simple Winchester, Dec.'86 Green.
Disk-31 *** Read/Write MS/PC-DOS (SK*DOS)
Disk-32 Hier-UNIX Type upgrade - 68MJ 2/87

NOTE:
This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

Specify 8" disk $19.50
5" disk $16.95

Add: S/H - $3.50
Overseas add: $4.50 surface - $7.00 Air Mail, USA Dollars

68 MICRO JOURNAL
PO Box 849
Hixson, TN 37343
615 842-4600 - Telex 510 600-6630

GMX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

GMX MICRO 20 PRICE LIST

MICRO 20 (12.5 MHz) W/1 SAB.....$2565.00
MICRO 20 (16.67 MHz) W/1 SAB.....$2895.00
MICRO 20 (20 MHz) W/1 SAB.....$3295.00

OPTIONAL PARTS AND ACCESSORIES

68881RC12.....$ 295.00
68881RC16.....$ 395.00
MOTOROLA 68020 USERS MANUAL.....$ 18.00
MOTOROLA 68881 USERS MANUAL.....$ 18.00

SBC ACCESSORY PACKAGE (M20-AP).....$1690.00
The package includes a PC-style cabinet with a custom backpanel, a Xebec 1410A hard disk controller, a 25 Megabyte (unformatted) hard disk, a 5 1/4" DSDD 80 track floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches and all necessary internal cabling
BACK PANEL PLATE (BPP-PC) For Above.....$ 44.00
2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD..$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD.....$ 780.00
TO SUBSTITUTE 65MB HD FOR 25MB HD, ADD....$1800.00

I/O EXPANSION BOARDS

16 PORT SERIAL CARD ONLY (SBC-16S).....$335.00
The SBC-16S e tends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 Adapter(SAB-25,SAB-9D or RJ-45).....$165.00
The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O CARD (SBC-60P).....$398.00
The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW).....$75.00
The S -WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA).....$195.00
The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN)....$475.00
The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coa ial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN.....$ 120.00

GMX MICRO 20 SOFTWARE

020 BUG UPDATE - PROMS & MANUAL.....$ 150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS.

OS9

OS9/68020 PROFESSIONAL PAK.....$ 850.00
Includes O.S., "C", EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.

Other Software for OS-9/68020

BASIC (included in PERSONAL PAK).....$ 200.00
C COMPILER (included in PROFESSIONAL PAK).$ 500.00
PASCAL COMPILER.....$ 500.00

UNIFLEX

UniFLEX (when ordered with Board).....$ 450.00
UniFLEX WITH REAL-TIME ENHANCEMENTS.....$1000.00

Other Software for UniFLEX

UniFLEX BASIC W/PRECOMPILER.....$ 300.00
UniFLEX C COMPILER.....$ 350.00
UniFLEX COBOL COMPILER.....$ 750.00
UniFLEX SCREEN EDITOR.....$ 150.00
UniFLEX TEXT PROCESSOR.....$ 200.00
UniFLEX SORT/MERGE PACKAGE.....$ 200.00
UniFLEX VSAM MODULE.....$ 100.00
UniFLEX UTILITIES PACKAGE I.....$ 200.00
UniFLEX PARTIAL SOURCE LICENSE.....$1000.00

GMX EXCLUSIVE VERSIONS,CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX).....$1500.00

SCULPTOR (specify UniFLEX or OS9).....$ 995.00

FORTH (OS9).....$ 595.00

DYNACALC (specify UniFLEX or OS9).....$ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURES PRODUCT.

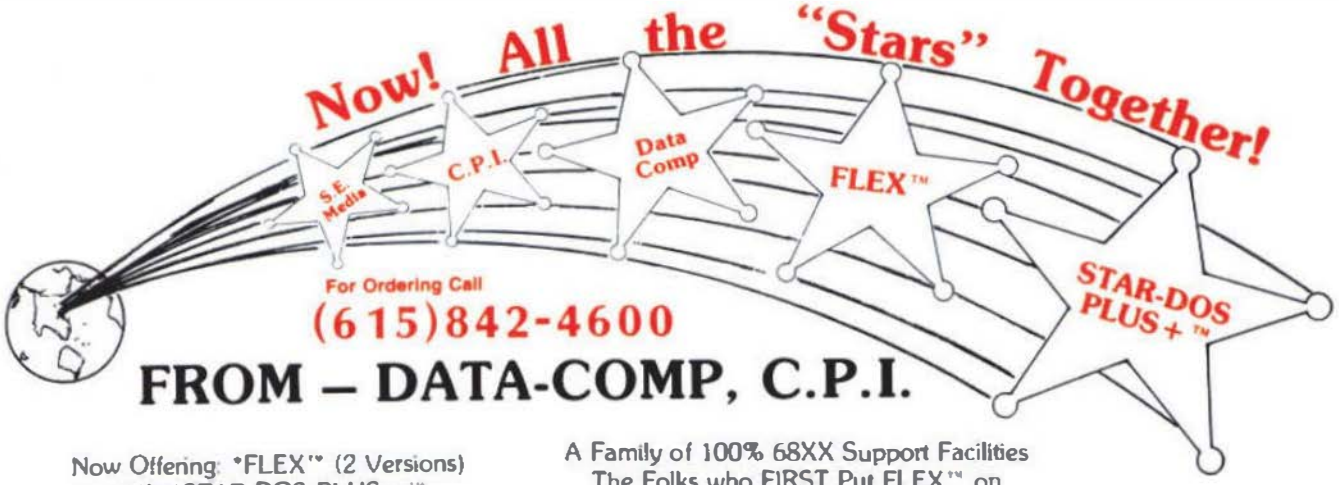
ALL PRICES ARE F.O.B. CHICAGO

GMX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 ndling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank acco nt in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago,IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

S - 50 BUS / 68XX

Board and/or Computer
Terminals-CRTs-Printers
Disk Drives-etc.

REPAIRS

This

Not This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to **'ALL'** S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. *Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and *confirm a Data-Comp service & shipping number*! This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but **NO** advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates *must be requested*. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with the item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630